



WavePort ActiveX Component User Manual

Version 1.1

REVISIONS HISTORY

Revision	Description	Author	Date	Comments
0	Original document	RCS	20/01/05	Version 1
1	Correction repeater format	RCS	27/01/05	Version 1

ACTIVEX VERSIONS

Wavenis.System component **V 1.0.0.1**

Wavenis.Waveport component **V 1.0.0.1**

➤ FIRMWARE VERSIONS SUPPORTED

WavePort / WaveCard : **Compatible with previous versions until the version 2.01**

Is)XPort : **Compatible with all the versions, except for the following functions which are not supported:**

- **GetSwitchMode**
- **SetSwitchMode**
- **GetErrorStatus**
- **SetErrorStatus**
- **RadioPacket**

TABLE OF CONTENTS

1 PRESENTATION OF THE WAVENIS.NET ACTIVEX.....	5
2 INTEGRATION OF THE WAVENIS.NET ACTIVEX UNDER VISUAL STUDIO.....	6
2.1 UNDER VISUAL BASIC 6.....	6
2.2 UNDER VISUAL C++ 6.....	8
3 INTEGRATION OF THE WAVENIS.NET ACTIVEX UNDER VISUAL STUDIO.NET.....	9
4 INITIALIZATION OF THE WAVENIS ACTIVEX UNDER VISUAL STUDIO 6.....	10
4.1 INITIALIZATION, AND OPENING OF THE COM PORT	10
4.2 EVENTS MANAGEMENT.....	11
4.2.1 Definition and polling of the event.....	11
4.2.2 Processing of the event.....	11
5 INITIALIZATION OF THE WAVENIS ACTIVEX UNDER VISUAL STUDIO .NET.....	12
5.1 INITIALIZATION, AND OPENING OF THE COM PORT	12
5.2 EVENTS MANAGEMENT.....	13
5.2.1 Connection to an event handler.....	13
5.2.2 Periodic scanning.....	13
5.2.3 Processing of the event.....	13
6 SPECIFIC FUNCTIONS TO THE WAVEPORT EQUIPMENTS.....	14
6.1 CONFIGURATION FUNCTIONS OF THE INTERNAL PARAMETERS.....	14
6.1.1 GetWakeUp : Reading of the polling period of RF medium radio.....	14
6.1.2 SetWakeUp : modification of the polling period of RF medium radio.....	14
6.1.3 GetWakeUpLong : reading of the WakeUp type to be used.....	15
6.1.4 SetWakeUpLong : selection of the WakeUp type to be used.....	15
6.1.5 GetWakeUpLength : reading of the length of Long WakeUp.....	16
6.1.6 SetWakeUpLength : modification of the length of Long WakeUp.....	16
6.1.7 GetGroupNumber : reading of the group number to be interrogated, in Polling mode.....	17
6.1.8 SetGroupNumber : modification of the group number to be interrogated, in Polling mode.....	17
6.1.9 GetRadioAck : reading of the radio acknowledgement configuration.....	18
6.1.10 SetRadioAck : modification of the radio acknowledgement configuration.....	18
6.1.11 GetPollingTable : reading of the addresses table used in polling mode.....	19
6.1.12 SetPollingTable : modification of the addresses table used in polling mode.....	19
6.1.13 GetPollingTime : reading of the delay between two consecutive emissions in radio polling mode.....	20
6.1.14 SetPollingTime : modification of the delay between two consecutive emissions in radio polling mode.....	20
6.1.15 GetFirmware : reading of the firmware version of the WaveCard/ WavePort.....	21
6.1.16 GetRadioAddress : reading of the radio address of the WaveCard/ WavePort.....	21
6.1.17 GetSwitchMode : reading of the SWITCH_MODE_STATUS parameter configuration.....	22
6.1.18 SetSwitchMode : configuration of the SWITCH_MODE_STATUS parameter.....	22
6.1.19 GetExchangeStatus : reading of the EXCHANGE_STATUS parameter configuration.....	23

6.1.20 SetExchangeStatus : configuration of the EXCHANGE_STATUS parameter.....	23
6.2 FUNCTIONS RELATED TO MANAGEMENT OF A COM PORT.....	24
6.2.1 Init : Initialization of the internal parameters, and opening of the COM port selected.....	24
6.2.2 GetCommPortNumber : reading of the selected COM port.....	24
6.2.3 SetCommPortNumber : selection of the COM port to be used.....	25
6.2.4 Open : Opening of the selected COM port.....	25
6.2.5 Close : close the application, and the COM port.....	26
6.3 FUNCTIONS RELATED TO EVENTS MANAGEMENT.....	27
6.3.1 ResetAlarm : acknowledgement of an alarm.....	27
6.3.2 WatchLink : detection and recovery of events.....	28
6.3.3 GetAddress : Read the radio address of the equipment which transmitted the alarm.....	28
6.3.4 GetStatus : Fetch the status of the alarm.....	29
6.3.5 GetTime : Read the date and time of the alarm	29
6.3.6 GetDayOfWeek : Read the day of week of the alarm detection.....	29
6.3.7 GetValue : fetch data from a received alarm frame.....	30
6.3.8 GetSensorType : read the equipment type of the module which has initiated the alarm	30
6.3.9 GetSensor : read the equipment type of the module which has initiated the alarm (string).....	31
6.4 FUNCTIONS RELATED TO THE RADIO EXCHANGES.....	32
6.4.1 RadioExchange : send a radio frame, in 'Frame Exchange' mode.....	33
6.4.2 RadioMessage : send a radio frame, in 'Message' mode.....	34
6.4.3 RadioPacket : transmit a frame with waiting of a response in 'multi-frame' mode.....	35
6.4.4 RadioExchangePolling : send a radio frame using the 'polling' mode.....	36
APPENDIX 1 – VB6 EXAMPLE CODE.....	37
APPENDIX 2 – VC++6 EXAMPLE CODE.....	39
APPENDIX 3 - VB.NET EXAMPLE CODE.....	48
APPENDIX 4 - C# .NET EXAMPLE CODE FOR WIN32.....	52
APPENDIX 5 - C# .NET EXAMPLE CODE FOR WINCE.....	58

1 PRESENTATION OF THE WAVENIS.NET ACTIVEX

This document presents the Wavenis.Net activeX component, and its integration in user applications. This ActiveX supports the following environments :

- ◆ Visual Studio.NET environment (C#.net ; VB.net ; C++.net ; ASP.net ; J#.net)
- ◆ Visual Studio 6 environment (VC++ 6 ; VB6)

But it can be also used under the environments Borland C++ Builder; Borland Delphi; LabView; and generally, under any environment supporting dotNet ActiveX.



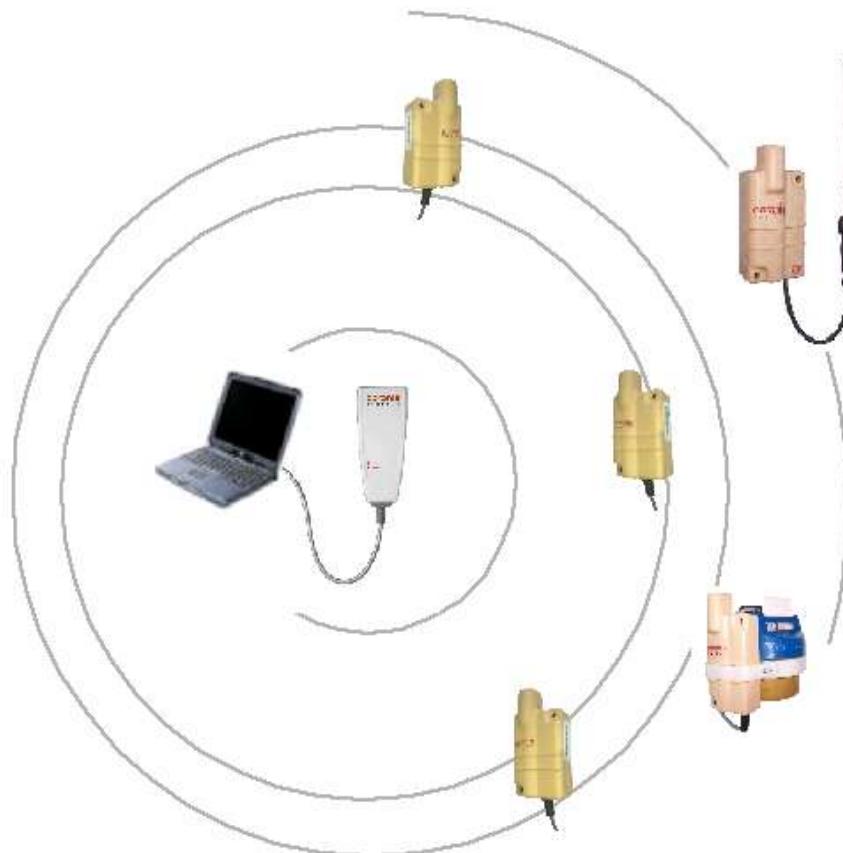
Compatibility :

- under Win32 : Windows 2000 ; Windows Millennium ; Windows 2003 server ;
Windows XP (Pro, et Family edition);
- under WinCE : Pocket PC 2002 ; Pocket PC 2003 ; Windows CE 4.2 ; Windows CE 5.0

The activeX is not compatible with WinCE 3.0, and Windows 98.

The activeX Wavenis.Net is composed:

- ◆ With **Waveport** component allowing to initialize the dialogue between the host, and WavePort (or WaveCard); and to communicate with one (or several) distant equipment.



Attention this handbook presents only the use of the ActiveX.Net component, and the detail of the associated functions. To have information about the principles of operation, and the formats of the data, it will imperatively be necessary to refer to the application handbook of the equipments concerned.

2 INTEGRATION OF THE WAVENIS.NET ACTIVEX UNDER VISUAL STUDIO

This chapter describes the integration method of Wavenis.Net ActiveX in Visual Studio environment.

The DLL Wavenis.Waveport provides a COM interface (IWavePort) which allows to use .NET DLL in an user application VB6 and C++6.

The COM component is associated a file (Wavenis.Waveport.tlb) which allows Visual BASIC 6 to refer the component ; and Visual C++ 6 to generate the proxy class interface IWavePort.

2.1 UNDER VISUAL BASIC 6

➤ Creation of a new project :

Select menu *File , New project, EXE Standard.*

➤ ActiveX integration :

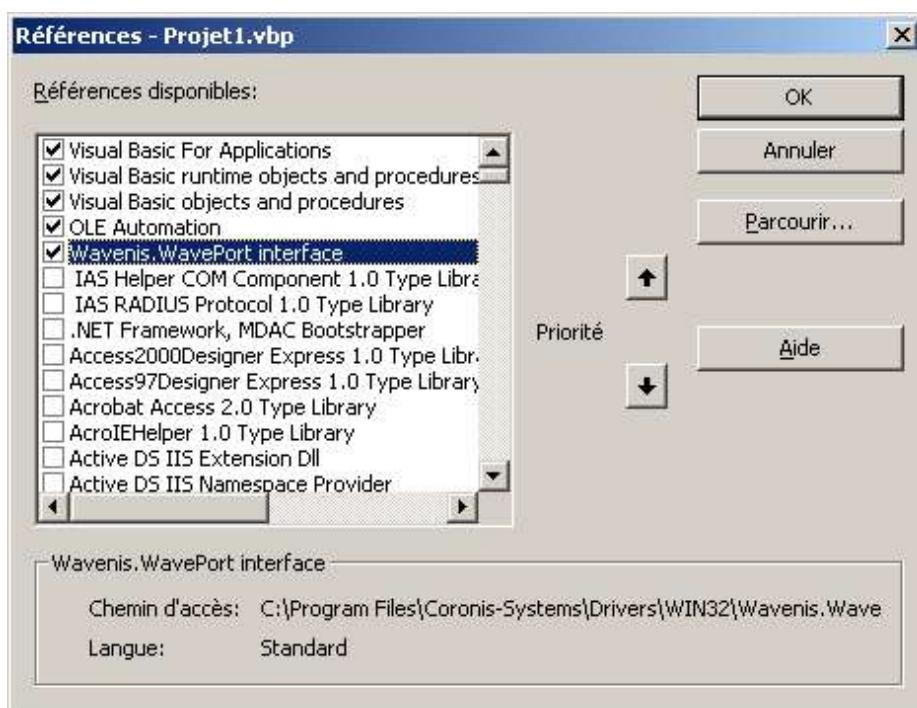
- ◆ under Visual Basic 6, select menu **Project – References**.
- ◆ Select the **Browse** button, and select the file **Wavenis.Waveport.tlb**.



Make sure to choose the folder corresponding to the environment in which the activeX will be used :

- under Win32 : C:\Program Files\Coronis-Systems\Drivers\WIN32\
- under WinCE : C:\Program Files\Coronis-Systems\Drivers\WINCE\

- ◆ In the available references, please check the new option "**Wavenis.WavePort interface**".



- ◆ Select menu *View – Objects explorer*, and choose the library **Wavenis_Waveport**

The Wavenis_Waveport library contains the following objects :

- Com component interface **IWavePort** to use under VB6;
- .NET component **WavePort**;
- alarm management : **alarmHandler**;
- event object : **IWavenisEvent** ;
- object containing enumeration of error codes : **reason**.

VB6**Visual Basic 6 Example :**

```
'Declaration of the interfaces
Dim WPort1 As IWavePort

' Creation of the Waveflow object
Set WPort1 = New Wavenis_WavePort

' Initialization of the WavePort object to the RS232 communication port COM1
Dim bSuccess As Boolean
'Initialize the WPort1
bSuccess = WPort1.Init(9600, 8, 0, 1, 0)

'Obtaining the firmware version of the Waveflow module whose radio address is "011604303220" :
Dim firmware As String
Dim rep As String
Dim strError As String

firmware = WPort1.RadioExchange("011604303220", "28", rep, strError)
```

2.2 UNDER VISUAL C++ 6

➤ Creation of a new project :

Select menu *File* , *New project*, *EXE Standard*.

➤ ActiveX integration :

Contrary to the Microsoft Visual BASIC clients who can post a .NET object in the objects explorer, thus exposing the object functions, syntax, properties and the fields, exactly as if it acted of any other object COM. The importation process of types library is slightly more complicated for the C++ clients, even if you use the same tools to export "metadata" towards a library of COM types.

To refer .NET members objects starting from a C++ client not managed, refer file .TLB with directive **#import** (see C++ example in appendix 2, line 12)



Make sure to choose the folder corresponding to the environment in which the activeX will be used :

- under Win32 : C:\Program Files\Coronis-Systems\Drivers\WIN32\
- under WinCE : C:\Program Files\Coronis-Systems\Drivers\WINCE\

3 INTEGRATION OF THE WAVENIS.NET ACTIVEX UNDER VISUAL STUDIO.NET

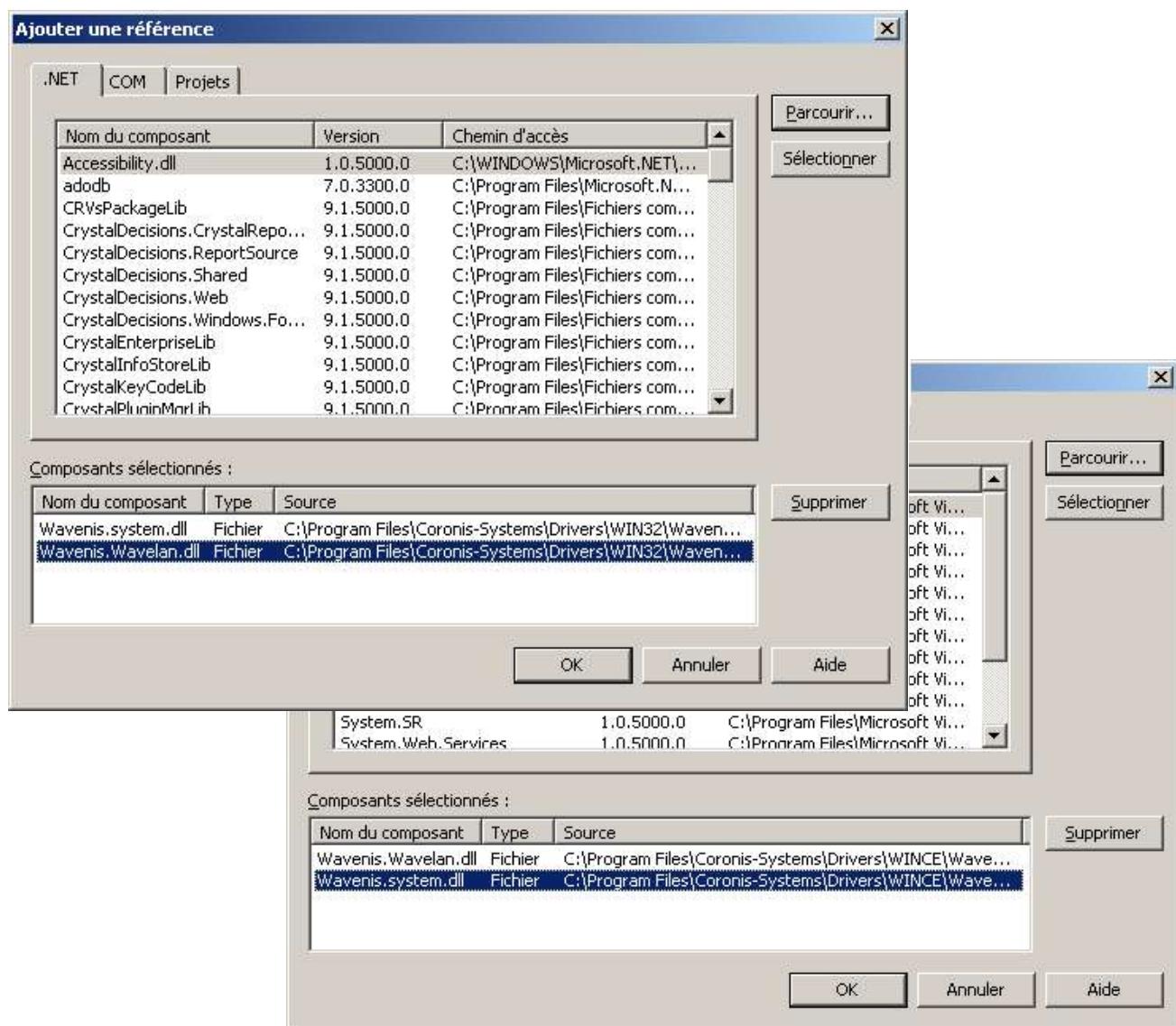
This chapter describes the method to integrate ActiveX Wavenis.Net in the Visual Studio.NET environment. The same method is applied whatever the language used (C#.net; VB.net; C++.net; ASP.net; J#.net)

➤ **Creation of a new project :**

Select menu *File* , *New project*, *EXE Standard*.

➤ **ActiveX integration :**

Select menu *Project*, then *Add a reference*.



Make sure to choose the folder corresponding to the environment in which the activeX will be used :

- under Win32 : C:\Program Files\Coronis-Systems\Drivers\WIN32\
- under WinCE : C:\Program Files\Coronis-Systems\Drivers\WINCE\

4 INITIALIZATION OF THE WAVENIS ACTIVEX UNDER VISUAL STUDIO 6

The initialization of the Wavenis.Net activeX is composed at least of the following parts :

- ◆ Opening and initialization of the COM port,
- ◆ Definition, and connection of the events manager.

Examples codes illustrate in the various supported languages, the implementation of the activex. They are available in appendix; and under the following folder :

C:\Program Files\Coronis-Systems\Wavenis.net\Applications\Win32
or C:\Program Files\Coronis-Systems\Wavenis.net\Applications\WinCE

4.1 INITIALIZATION, AND OPENING OF THE COM PORT

First of all, it is necessary to define the WavePort object, and to create its instance before being able to use it. The definitions of the object, and its instance are illustrated in appendix 1, and appendix 2 :

- ◆ under Visual Basic 6, lines 1, and 5

1	Dim WPort As IWavePort
5	Set WPort = New Wavenis_Waveport.WavePort

- ◆ under Visual C++6, line 42, and from line 357 to 374

42	Wavenis_Waveport::IWAVEPORTptr Wport;
370	hr = ::CoCreateInstance(CLSID_WavePort,
371	NULL,
372	CLSCtx_INPROC_SERVER,
373	IID_IWAVEPORT,
374	reinterpret_cast<void**>(&WPort));

Then, open the COM port COM by using the following functions,

- **SetCommPortNumber** (port number) : *This function selects the COM port to be used for the communication between WavePort and its host. See details in chapter 6.2.3.*
- **bool Init** (baudrate, databits, parity, stopbit, mask) : *configure, and opens the port COM selected with SetCommPortNumber. See details in chapter 6.2.1.*

Examples of initialization, and opening of the COM port are available in VB6, and VC++6 in appendix 1, and 2 :

- ◆ under VB6, lines 30, and 33

30	WPort.SetCommPortNumber (iCom)
33	bSuccess = WPort.Init(9600, 8, 0, 1, 0)

- ◆ under VC++6, lines 277, and 280

277	hr = WPort->SetCommPortNumber(iCom);
280	hr = WPort->Init(9600,8,0,1,0,&bSuccess);

4.2 EVENTS MANAGEMENT

WavePort (or WaveCard) is likely to receive spontaneous messages from distant modules. To recover these messages under user application, it is necessary to create an events manager.

In Visual Studio 6 environment, the events manager is controlled by a timer scanning the serial connection periodically, in waiting of a possible message (see example of code in appendix 1, and 2).

When the WavePort module receive an event, and send it to its host, it awaits a RS232 acknowledgement from the host. If the acknowledgement does not arrive at the end of 500ms, then the module re-emits the event (see manual application of the WavePort modules).

The scanning controlled by a timer must be carried out every 200ms (230ms max) so that the user application has time to send the acknowledgement to the initiator equipment of the event. The alarm acknowledgement is sent by using *ResetAlarm* function. (see chapter 6.3.1).



Attention, currently when the initiator equipment of the event is accessible via repeaters. It is possible to acknowledge the event only if the repeater route is known by the user; else the acknowledgement is impossible.

This is due to the fact that the repeater route is not contained in the alarm frame.

In VB6, when creating the timer, the user has to disable the timer starts in the timer configuration box. It will be enable by the user, when creating the instance of the WavePort object.



4.2.1 Definition and polling of the event

In the periodic function of the timer, an event object is declared according to the **Wavenis_Waveport.IWavenisEvent** class. Then, this periodic function make a call to the **WatchLink** function which will check the arrival of an event; and will store it in the event object created (WEVENT). See the detail of the **WatchLink** function in chapter 6.3.2 and examples code en appendix 1, and 2 :

- under VB6, lines 64, and 66

64	Dim WEvent As Wavenis_Waveport.IWavenisEvent
66	If (WPort.WatchLink(WEVENT)) Then

- under C++6, lines 391, and 393

391	Wavenis_Waveport::IWavenisEventPtr WEvent=NULL;
393	HRESULT hr = WPort->WatchLink(&WEVENT,&result);

4.2.2 Processing of the event

When an event was detected by the WatchLink function, the event is stored in the instance of the **IWavenisEvent** object.

Consequently, the user has access to a certain number of information about the event, accessible via the following functions: **GetAddress**; **GetStatus**; **GetTime**; **GetDayOfWeek**; **GetValue**; **GetSensorType**; **GetSensor**. The detail of these functions is described in chapter 6.3.

5 INITIALIZATION OF THE WAVENIS ACTIVEX UNDER VISUAL STUDIO.NET

The initialization of the Wavenis.Net activeX is composed at least of the following parts :

- ◆ Opening and initialization of the COM port,
- ◆ Definition, and connection of the events manager.

Examples codes illustrate in the various supported languages, the implementation of the activex. They are available in appendix; and under the following folder :

C:\Program Files\Coronis-Systems\Wavenis.net\Applications\Win32
or C:\Program Files\Coronis-Systems\Wavenis.net\Applications\WinCE

5.1 INITIALIZATION, AND OPENING OF THE COM PORT

First of all, it is necessary to define the WavePort object, and to create its instance before being able to use it. The definitions of the object, and its instance are illustrated in appendix 3, 4 and 5 :

- ◆ under VB.net, lines 3, and 14 :

3	Dim WPort As Wavenis.WavePort
14	WPort = New Wavenis.WavePort

- ◆ under VC#.net, lines 7, 21, and 39 :

7	using Wavenis;
21	private Wavenis.WavePort WPort;
39	WPort = new WavePort();

Then, open the COM port by using the following functions,

- **SetCommPortNumber** (port number) : *This function selects the COM port to be used for the communication between WavePort and its host. See details in chapter 6.2.3.*
- **bool Init** (baudrate, databits, parity, stopbit, mask) : *configure, and opens the port COM selected with SetCommPortNumber. See details in chapter 6.2.1.*

Examples of initialization, and opening of the COM port are available in VB.net, and VC#.net in appendix 3, 4, and 5.

- ◆ under VB.net, lines 149, and 152 :

149	WPort.SetCommPortNumber(iCom)
152	If (WPort.Init(9600, 8, 0, 1, 0)) Then

- ◆ under VC#.net, lines 195, and 198 :

195	WPort.SetCommPortNumber(sCom);
198	bool bresult = WPort.Init(9600,8,0,1,0);

5.2 EVENTS MANAGEMENT

WavePort (or WaveCard) is likely to receive spontaneous messages from distant modules. To recover these messages under user application, it is necessary to create an events manager.

The events management is done by two methods, applied according to the language used :

- ◆ connection of the events to an handler sending the event to the application. This method is applicable under VB.Net; and VC#.Net.
- ◆ Use of a periodic function scanning the serial port, and waiting for an event. This method is used with the VC++.Net language

The VC++.Net language being very close to the VC++, it also uses the events management per periodic scanning.

When the WavePort module receive an event, and send it to its host, it awaits a RS232 acknowledgement from the host. If the acknowledgement does not arrive at the end of 500ms, then the module re-emits the event (see manual application of the WavePort modules).



Attention, currently when the initiator equipment of the event is accessible via repeaters. It is possible to acknowledge the event only if the repeater route is known by the user; else the acknowledgement is impossible.

This is due to the fact that the repeater route is not contained in the alarm frame.

5.2.1 Connection to an event handler

The event class IWavenisEvent contains the data of the alarm transmitted by the distant module. This class provides a delegate 'alarmHandler' to connect the alarm manager of Waveport to a function defined by the application which will receive the event.

When using an event handler, the application comes to scan periodically the serial port, automatically every 230ms. If a spontaneous message is detected, the function 'alarmRang' is called.

Connection to the handler is done in the following way :

- ◆ under VB.Net : by the use of the function 'AddHandler' (appendix 3, line 155) :

155 AddHandler WPort.Alarm, AddressOf Me.alarmRang

- ◆ under VC# .Net : by the creation of an event object 'alarmhandler' (appendix 4, line 205)

205 WPort.Alarm += new Wavenis.alarmHandler(alarmRang);

5.2.2 Periodic scanning

The events manager is controlled by a timer scanning the serial connection periodically, in waiting of a possible message.

The scanning controlled by a timer, must be carried out every 200ms (230ms max) so that the user application has time to send the acknowledgement to the initiator equipment of the event. The alarm acknowledgement is sent by using *ResetAlarm* function. (see chapter 6.3.1).

In the periodic function of the timer, an event object is declared according to the **Wavenis_Waveport.IWavenisEvent** class. Then, this periodic function make a call to the **WatchLink** function which will check the arrival of an event; and will store it into the event object created (WEvent).

5.2.3 Processing of the event

Whatever the event management method used, when an event was detected, it is stored in the instance of the IWavenisEvent object.

Consequently, the user has access to a certain number of information about the event, accessible via the following functions: GetAddress; GetStatus; GetTime; GetDayOfWeek; GetValue;GetSensorType; GetSensor. The detail of these functions is described in chapter 6.3.

6 SPECIFIC FUNCTIONS TO THE WAVEPORT EQUIPMENTS

6.1 CONFIGURATION FUNCTIONS OF THE INTERNAL PARAMETERS

For the detail of operation relative to each parameter, please refer to the application handbook of the WavePort (WaveCard).

6.1.1 GetWakeUp : *Reading of the polling period of RF medium radio*

VB6GetWakeUp (**(out)** **wakeUpInterval** as Integer) as Boolean**C# .Net**bool GetWakeUp (ref int **wakeUpInterval**)

➤ Parameters :

none

➤ Returned values :

wakeUpInterval : polling period of RF medium radio, in multiples of 100ms**bResult** : returned value 'OK' = 1
 'NOK' = 0

6.1.2 SetWakeUp : *modification of the polling period of RF medium radio*

VB6SetWakeUp (**(in)** **wakeUpInterval** as Short) as Boolean**C# .Net**bool SetWakeUp (short **wakeUpInterval**)

➤ Parameters :

wakeUpInterval : polling period of RF medium radio, in multiples of 100ms
By default := 10 , for 1 seconds (if = 0, then quasi-permanent reception)

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.3 GetWakeUpLong : reading of the WakeUp type to be used

This function allows to read the WakeUp type to be used to transmit a frame.

VB6

```
GetWakeUpLong (      (out) wakeUpLong as Boolean ) as Boolean
```

C# .Net

```
bool GetWakeUpLong (ref bool wakeUpLong)
```

➤ Parameters :

none

➤ Returned values :

wakeUpLong : 0 : Long Wake Up (default value)
1 : Short Wake Up

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.4 SetWakeUpLong : selection of the WakeUp type to be used

This function allows to configure the WakeUp type to be used to transmit a frame.

VB6

```
SetWakeUpLong (      (in) wakeUpLong as Boolean ) as Boolean
```

C# .Net

```
bool SetWakeUpLong (bool wakeUpLong)
```

➤ Parameters :

wakeUpLong : 0 : Long Wake Up (default value)
1 : Short Wake Up

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.5 GetWakeUpLength : *reading of the length of Long WakeUp.*

VB6 | GetWakeUpLength ((**out**) **wakeUpLength** as Short) as Boolean

C# .Net | bool GetWakeUpLength (ref short **wakeUpLength**)

➤ Parameters :

none

➤ Returned values :

wakeUpLength : duration of the Long WakeUp in multiples of 1ms.

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.6 SetWakeUpLength : *modification of the length of Long WakeUp.*

This method modifies the length of Long WakeUp, when this one is used. In general, it is equal to the polling period of RF medium radio + 100ms.

VB6 | SetWakeUpLength ((**in**) **wakeUpLength** as Short) as Boolean

C# .Net | bool SetWakeUpLength (short **wakeUpLength**)

➤ Parameters :

wakeUpLength : duration of the Long WakeUp in multiples of 1ms.

Default value : = 1100 ms (min. = 20ms ; max. = 10 sec)

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.7 GetGroupNumber : reading of the group number to be interrogated, in Polling mode

VB6GetGroupNumber (**(out)** **groupNumber** as Short) as Boolean**C# .Net**bool GetGroupNumber (ref short **groupNumber**)

➤ Parameters :

none

➤ Returned values :

groupNumber : group number to be interrogated**bResult** : returned value 'OK' = 1
 'NOK' = 0

6.1.8 SetGroupNumber : modification of the group number to be interrogated, in Polling mode

This function modifies the group number to be interrogated on selective Polling request (refer to the parameter 0x09 in the application handbook of WaveCard/ WavePort).

VB6SetGroupNumber (**(in)** **groupNumber** as Short) as Boolean**C# .Net**bool SetGroupNumber (short **groupNumber**)

➤ Parameters :

groupNumber : group number to be interrogated

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.9 GetRadioAck : reading of the radio acknowledgement configuration

VB6GetRadioAck (**(out)** **bRadioAck** as Boolean) as Boolean**C# .Net**bool GetRadioAck (ref bool **bRadioAck**)

➤ Parameters :

none

➤ Returned values :

bRadioAck

: radio acknowledgement configuration

*0 : no radio acknowledgement (default value)
1 : radio acknowledgement activated***bResult**

: returned value

'OK' = 1
'NOK' = 0

6.1.10 SetRadioAck : modification of the radio acknowledgement configuration

VB6SetRadioAck (**(in)** **bRadioAck** as Boolean) as Boolean**C# .Net**bool SetRadioAck (bool **bRadioAck**)

➤ Parameters :

bRadioAck

: radio acknowledgement configuration

*0 : no radio acknowledgement (default value)
1 : radio acknowledgement activated*

➤ Returned values :

bResult

: returned value

'OK' = 1
'NOK' = 0

6.1.11 GetPollingTable : reading of the addresses table used in polling mode

This function allows to read the table containing the radio addresses of the distant equipments having to be interrogated in Polling mode (refer to the parameter 0x08 in the application handbook WaveCard/ WavePort). The maximum number is of 40 equipments being able to be interrogated.

VB6

GetPollingTable (**(out) pollingTable** as String) as Boolean

C# .Net

bool GetPollingTable (ref string **pollingTable**)

➤ Parameters :

none

➤ Returned values :

pollingTable

: List containing the radio addresses of the equipments to interrogated
1st byte: number of recipients (max = 40)
following bytes : radio addresses of the recipient (6 bytes per address)

bResult

: returned value 'OK' = 1
'NOK' = 0

6.1.12 SetPollingTable : modification of the addresses table used in polling mode

This function allows to modify the table containing the radio addresses of the distant equipments having to be interrogated in Polling mode (refer to the parameter 0x08 in the application handbook WaveCard/ WavePort). The maximum number is of 40 equipments being able to be interrogated.

VB6

SetPollingTable (**(in) pollingTable** as String) as Boolean

C# .Net

bool SetPollingTable (string **pollingTable**)

➤ Parameters :

pollingTable

: List containing the radio addresses of the equipments to interrogated
1st byte: number of recipients (max = 40)
following bytes : radio addresses of the recipient (6 bytes per address)

➤ Returned values :

bResult

: returned value 'OK' = 1
'NOK' = 0

6.1.13 GetPollingTime : *reading of the delay between two consecutive emissions in radio polling mode*

This function reads the value of the delay separating two consecutive emissions, in radio polling mode (refer to the parameter 0x0A in the application handbook of the WaveCard/ WavePort).

VB6

```
GetPollingTime ( (out) pollingTime as Short ) as Boolean
```

C# .Net

```
bool GetPollingTime ( ref short pollingTime )
```

➤ Parameters :

none

➤ Returned values :

pollingTime : time separating two consecutive emissions, in multiple of 100ms
by default PollingTime = 10 , for a delay of 1 second.

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.14 SetPollingTime : *modification of the delay between two consecutive emissions in radio polling mode*

This function configures the delay separating two consecutive emissions, in radio polling mode (refer to the parameter 0x0A in the application handbook of the WaveCard/ WavePort).

VB6

```
SetPollingTime ( (in) pollingTime as Short ) as Boolean
```

C# .Net

```
bool SetPollingTime (short pollingTime)
```

➤ Parameters :

pollingTime : time separating two consecutive emissions, in multiple of 100ms
by default PollingTime = 10 , for a delay of 1 second.

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.15 GetFirmware : reading of the firmware version of the WaveCard/ WavePort

VB6

GetFirmware (**(out) firmware** as String) as Boolean

C# .Net

bool GetFirmware (ref string **firmware**)

➤ Parameters :

none

➤ Returned values :

firmware	: firmware version of the equipment (on 5 bytes : [B4:B0]) byteB4 : character 'V' in ASCII format bytes [B3:B2] : radio transmission mode of the equipment bytes [B1:B0] : firmware version
bResult	: returned value 'OK' = 1 'NOK' = 0

6.1.16 GetRadioAddress : reading of the radio address of the WaveCard/ WavePort

VB6

GetRadioAddress (**(out) radioAddress** as String) as Boolean

C# .Net

bool GetRadioAddress (ref string **radioAddress**)

➤ Parameters :

none

➤ Returned values :

radioAddress	: radio address of the WaveCard / WavePort
bResult	: returned value 'OK' = 1 'NOK' = 0

6.1.17 GetSwitchMode : reading of the SWITCH_MODE_STATUS parameter configuration

When the SWITCH_MODE_STATUS is activated, the WavePort automatically selects the transmission mode of a radio request, according to the address of the recipient.

VB6 GetSwitchMode (**(out)** switchMode as Boolean) as Boolean

C# .Net bool GetSwitchMode (ref bool **switchMode**)

➤ Parameters :

none

➤ Returned values :

switchMode : 0 : automatic selection deactivated (default value)
 1 : automatic selection activated

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.18 SetSwitchMode : configuration of the SWITCH_MODE_STATUS parameter

When the SWITCH_MODE_STATUS is activated, the WavePort automatically selects the transmission mode of a radio request, according to the address of the recipient.

VB6 GetSwitchMode (**(in)** switchMode as Boolean) as Boolean

C# .Net bool SetSwitchMode (bool **switchMode**)

➤ Parameters :

switchMode : 0 : automatic selection deactivated (default value)
 1 : automatic selection activated

➤ Returned values :

bResult : returned value 'OK' = 1
 'NOK' = 0

6.1.19 GetExchangeStatus : reading of the EXCHANGE_STATUS parameter configuration

When parameter EXCHANGE_STATUS is activated, the WavePort send to its host a status message, if it encounters an error during transmission of a frame.

VB6

GetExchangeStatus (**(out)** exchangeStatus as Short) as Boolean

C# .Net

bool GetExchangeStatus (ref short exchangeStatus)

➤ Parameters :

none

➤ Returned values :

exchangeStatus : 0 : status message disabled (default value)
1 : status message enabled

bResult

: returned value 'OK' = 1
 'NOK' = 0

6.1.20 SetExchangeStatus : configuration of the EXCHANGE_STATUS parameter

When parameter EXCHANGE_STATUS is activated, the WavePort send to its host a status message, if it encounters an error during transmission of a frame.

VB6

SetExchangeStatus (**(in)** exchangeStatus as short) as Boolean

C# .Net

bool GetExchangeStatus (short exchangeStatus)

➤ Parameters :

exchangeStatus : 0 : status message disabled (default value)
1 : status message enabled

➤ Returned values :

bResult

: returned value 'OK' = 1
 'NOK' = 0

6.2 FUNCTIONS RELATED TO MANAGEMENT OF A COM PORT

6.2.1 Init : Initialization of the internal parameters, and opening of the COM port selected

Refresh the internal parameters of WavePort (or WaveCard) with the default values, and opens the port COM selected with the *SetCommPortNumber* function.

In the case of a second call to this method, the previous port is automatically closed, and the new COM port is opened.

VB6

```
Init ( (in) dwBaud as Short,
      (in) dataBits as Short,
      (in) parity as Short,
      (in) stopbit as Short,
      (in) mask as Integer ) as Boolean
```

C# .Net

```
bool Init (short dwBaud, short dataBits, short parity, short stopbit, int dcbMask )
```

➤ Parameters :

dwBaud	: serial baudrate,
dataBits	: number of data bytes,
parity	: 0 : no parity only value currently authorized , 1 : odd parity, 2 : even parity, 3 : 'mark' parity (parity bit always at 1), 4 : 'space' parity (parity bit always at 0)
stopbit	: number of stop bit,
dcbMask	: this mask allows to configure the signals of control of the serial connection. In standard operation, no control signal is used, so the mask is set to the value 0. For any other use, please consult us (technicalsupport@coronis.com).

➤ Returned values :

bResult	: returned value	'OK' = 1
		'NOK' = 0

6.2.2 GetCommPortNumber : reading of the selected COM port

This function returns the COM port number selected by *SetCommPortNumber* function.

VB6

```
GetCommPortNumber ( ) as Short
```

C# .Net

```
short GetCommPortNumber ()
```

➤ Parameters :

none

➤ Returned values :

sResult	: COM port number
----------------	-------------------

6.2.3 SetCommPortNumber : selection of the COM port to be used

VB6

SetCommPortNumber (**(in) portNumber** as Short) as void

C# .Net

void SetCommPortNumber (short **portNumber**)

➤ Parameters :

portNumber : COM port number to be used

➤ Returned values :

no value returned

6.2.4 Open : Opening of the selected COM port

This function opens the selected COM port by the *SetCommPortNumber* function. In the case of a second call to this method, the previous port is automatically closed.

Attention, contrary to the *Init* function, the *Open* function does not update the internal parameters with their default value.

VB6

Open (**(in) dwBaud** as Short,
(in) dataBits as Short,
(in) parity as Short,
(in) stopbit as Short,
(in) dcbMask as Integer) as Boolean

C# .Net

bool Open (short **dwBaud**, short **dataBits**, short **parity**, short **stopbit**, int **dcbMask**)

➤ Parameters :

dwBaud : serial baudrate,

dataBits : number of data bytes,

parity : 0 : no parity **only value currently authorized**,

1 : odd parity,

2 : even parity,

3 : 'mark' parity (parity bit always at 1),

4 : 'space' parity (parity bit always at 0)

stopbit : number of stop bit,

dcbMask : this mask allows to configure the signals of control of the serial connection. In standard operation, no control signal is used, so the mask is set to the value 0. For any other use, please consult us (technicalsupport@coronis.com).

➤ Returned values :

bResult : returned value 'OK' = 1
'NOK' = 0

6.2.5 Close : *close the application, and the COM port*

This function close the application, and close the COM port.

VB6

Close () as void

C# .Net

void Close ()

➤ Parameters :

none

➤ Returned values :

none

6.3 FUNCTIONS RELATED TO EVENTS MANAGEMENT

6.3.1 ResetAlarm : acknowledgement of an alarm

When the received spontaneous message is an alarm, it must be acknowledged by a call to this function. If the alarm is not acknowledged, then the distant equipment (initiator of the alarm) will send it again a certain number of times (please refer to the application handbook of the concerned equipment).



Attention, currently when the initiator equipment of the event is accessible via repeaters. It is possible to acknowledge the event only if the repeater route is known by the user; else the acknowledgement is impossible.

This is due to the fact that the repeater route is not contained in the alarm frame.

VB6

```
ResetAlarm ( (in) address as String,  

            (in) repeater as String,  

            (in) status as String,  

            (out) error as String ) as String
```

C# .Net

```
string ResetAlarm (string address, string repeater, string status, string error)
```

➤ Parameters :

- | | |
|-----------------|---|
| address | : radio address of the initiator of the spontaneous message |
| repeater | : list of the repeaters necessary to reach the remote module.
<i>format : string containing the number of repeaters to be used, and the addresses of the repeaters consecutively (without separator)</i>
<i>example : rep = "0301160430301D011604301D7D011604301D7E"</i>
'Nb rep Module 1 address Module 2 address Module 3 address ' |
| status | : status to be returned to acknowledge the alarm. Received alarm contains a status allowing to identify alarm; this status must be returned in the acknowledgement frame so that the initiator of the alarm identifies acknowledged alarm (example of several alarms transmission by the same module). |

➤ Returned values :

- | | |
|------------------------|--|
| error as String | : Return '0' if the command is processed successfully, if not returns a string containing the error message. |
|------------------------|--|



Example of 'error' values :

XML_GetInvalid =	"invalid Get method in the XML configuration file : "
XML_SetInvalid =	"invalid Set method in the XML configuration file : "
XML_ParamInvalid =	"invalid parameter in the XML configuration file : "
XML_ParamLengthInvalid =	"invalid parameter length in the XML configuration file : "
XML_GetUndefined =	"Get method not defined in the XML configuration file : "
XML_SetUndefined =	"Set method not defined in the XML configuration file : "
XML_ParamUndefined =	"parameter not defined in the XML configuration file : "
XML_ParamLengthUndefined =	"parameter length not defined in the XML configuration file : "
COM_RadioError =	"radio communication error"
COM_RS232Error =	"rs232 communication error"
COM_InvalidParamLength =	"invalid parameter length : "
COM_InvalidParamValue =	"invalid parameter value : "
COM_InvalidAddress =	"invalid address : "
COM_InvalidBufferSize =	"invalid buffer size : "

strResult

: no value returned (*reserved for future development*)

6.3.2 WatchLink : detection and recovery of events

This function must be used periodically to detect the arrival of a spontaneous message, and will fetch it if necessary.



Compatibility: This method should be implemented only in the case of an events management per periodic detection. Please refer to the chapters on the events management of the concerned language.

VB6

```
WatchLink (      (out) alarmEvent as IWavenisEvent ) as Boolean
```

C# .Net

```
bool WatchLink (Wavenis.IWavenisEvent alarmEvent)
```

➤ Parameters :

alarmEvent : instance of the IWavenisEvent object, used to store the alarm.

➤ Returned values :

bResult : returned value 'OK' = 1 (*alarm detected and fetched*)
'NOK' = 0

VB6

```
Private Sub Timer1_Timer()      ' Scrutation périodique d'événements
    Dim WEvent As Wavenis_Waveport.IWavenisEvent          ' déclaration d'un objet événement
    If ( Wport.WatchLink ( Wevent ) ) Then
        recv.Caption = WEvent.GetAddress + "=" + WEvent.GetValue
    End If
End Sub
```

When an alarm is detected and fetched to the host, the user is able to read pieces of information relating to the alarm, using the following functions :

6.3.3 GetAddress : Read the radio address of the equipment which transmitted the alarm

VB6

```
GetAddress ( ) as String
```

C# .Net

```
string GetAddress ()
```

➤ Parameters :

none

➤ Returned values :

strResult : radio address of the equipment which transmitted the alarm
(for example : "011603300373")

6.3.4 GetStatus : Fetch the status of the alarm

If the alarm is coming from a metering module (not from a WaveCard / WavePort), then it contains a status field used to identify the alarm type (see user manual of the concerned module).

This status is also used by the receiver of the alarm to acknowledge it. It allows the initiator to identify which alarm is acknowledged (if several alarm frame were transmitted).

VB6 | GetStatus () as String

C# .Net | string GetStatus ()

➤ Parameters :

none

➤ Returned values :

strResult : string containing the status of the alarm converted in decimal (the meaning of the status depends of the alarm, and the equipment type. Please refer to the user manual of the concerned equipment).
(for example "17" represent a status value = 0x11)

6.3.5 GetTime : Read the date and time of the alarm

VB6 | GetTime () as String

C# .Net | string GetTime ()

➤ Parameters :

none

➤ Returned values :

strResult : Date and time of the alarm (format : Date, followed by Time separate by a space character)



Example : "28/10/2004 21:52"

6.3.6 GetDayOfWeek : Read the day of week of the alarm detection

VB6 | GetDayOf Week () as Short

C# .Net | short GetDayOfWeek ()

➤ Parameters :

none

➤ Returned values :

sResult : day of week, of the alarm detection
0:Sunday 1:Monday 2:Tuesday 3:Wednesday

4:Thursday 5:Friday 6:Saturday

6.3.7 GetValue : fetch data from a received alarm frame

VB6

GetValue () as String

C# .Net

string GetValue ()

➤ Parameters :

none

➤ Returned values :

strResult : data from the alarm frame



Note : the fetched data depends of the equipment type which has initiated the alarm.

- Alarm coming from a WavePort (spontaneous message) : only the data following the transmitter radio address will be fetched

example : 300112043000010102030405, only data 0102030405 will be fetched.

- Alarm coming from a module (alarm frame): only the data relative to the cause of the alarm are fetched (the Date and Status fields are fetched by other functions)

example of a WaveFlow alarm: "A,9" corresponds to a water leakage on input A of 9 litres/hour if the module measures every hour.

6.3.8 GetSensorType : read the equipment type of the module which has initiated the alarm

VB6

GetSensorType () as String

C# .Net

string GetSensorType ()

➤ Parameters :

none

➤ Returned values :

strResult : value representing the equipment type.

for example, "22" represent a WaveFlow module (0x16)

6.3.9 GetSensor : *read the equipment type of the module which has initiated the alarm (string)***VB6**

GetSensorType () as String

C# .Net

string GetSensor ()

➤ Parameters :

none

➤ Returned values :

strResult : string containing the label of the equipment type*Example of returned values :*

- "AlarmWavePort"
- "AlarmWaveFlow"
- "AlarmWaveTherm"
- "AlarmWaveSense"
- "AlarmWaveTalk"
- "AlarmWaveCell"
- "AlarmWaveHub"
- "AlarmWaveLog"

6.4 FUNCTIONS RELATED TO THE RADIO EXCHANGES

The radio exchanges are composed of several modes of transmission/reception. With in certain cases, the possibility of receiving several consecutive radio frames (multi frames mode, accessible in reception only).

The following modes allow **point-to-point** exchanges :

- ◆ **'Frame Exchange' mode** : This mode allows to emit a request, with waiting of a radio response from the distant equipment.
- ◆ **'Message' mode** : allows to emit a request, without waiting of a radio response from the distant equipment.

Moreover exchanges of the point-to-point types have an additional mode which allow to reach a module out of radio range of the transmitter, by relaying the frames via other equipments.

- ◆ **'Relaying' mode** : this functionality allow to use a radio equipment to repeat a frame which is not initially intended to him.

The remaining modes allow exchanges with several distant equipments, in a selective way or not.

- ◆ **'Polling' mode** : allows to address a request to a list of known distant equipment. The response is sent to the host, transmitter of the request, when all the distant equipment responded, or on time-out.
- ◆ **'Broadcast' mode** : allows to address a request to all the distant equipment within radio range of the transmitter, or only to a group of equipment in radio range of the transmitter.

◆ **Particular Case: multi-frames reception**

Multi frame mode allows multi frame exchange between the Wavecard/Waveport (considered like the master of the exchange) and one of the telemetry equipments of the Coronis Systems product family (WaveTherm, WaveFlow, WaveSens, ...).

Current version of Wavecard does not allow multi frame mode between two Wavecard/Waveport equipments.



Remark : The current version of the ActiveX component does not allow to use the BroadCast communication mode. This functionality will be implemented in a future version.

6.4.1 RadioExchange : send a radio frame, in 'Frame Exchange' mode

this function allows to transmit a radio frame to a known distant equipment, with waiting of an answer (see WaveCard user manual).



Currently, in relaying mode, the relay route is not automatically sent to the distant equipment.
Consequently, the user must ensure that the relay route is known by the transmitter and the receiver of a relayed frame.

VB6

```
RadioExchange (      (in) address as String,
                    (in) frame as String,
                    (in) repeater as String,
                    (in) timeout as ulong,
                    (out) error as String ) as String
```

C# .Net

```
string RadioExchange (string address, string frame, string repeater, int timeout, string error)
```

➤ Parameters :

address frame repeater timeout	: radio address of the recipient (6 bytes) : data frame to transmit (max size : see remark below) : list of the repeaters necessary to reach the remote module. <i>format : string containing the number of repeaters to be used, and the addresses of the repeaters consecutively (without separator)</i> <i>example : rep = "0301160430301D011604301D7D011604301D7E"</i> <i>'Nb rep Module 1 address Module 2 address Module 3 address '</i>
---	---



Remark : maximum size definition

- Point to Point mode : **Max** = 152 bytes of data
- Relaying mode : **Max** = 152 - (2 + 6 x Number of repeaters)

=> 1 repeater : 144 bytes of data,
=> 2 repeaters : 138 bytes of data,
=> 3 repeaters : 132 bytes of data.

➤ Returned values :

strResult error	: awaited radio response. (<i>blank in case of error</i>) : = '0' if function succeeded; if not, returns a string containing an error message.
----------------------------------	--



Example of 'error' values :

XML_GetInvalid =	"invalid Get method in the XML configuration file : "
XML_SetInvalid =	"invalid Set method in the XML configuration file : "
XML_ParamInvalid =	"invalid parameter in the XML configuration file : "
XML_ParamLengthInvalid =	"invalid parameter length in the XML configuration file : "
XML_GetUndefined =	"Get method not defined in the XML configuration file : "
XML_SetUndefined =	"Set method not defined in the XML configuration file : "
XML_ParamUndefined =	"parameter not defined in the XML configuration file : "
XML_ParamLengthUndefined =	"parameter length not defined in the XML configuration file : "
COM_RadioError =	"radio communication error"
COM_RS232Error =	"rs232 communication error"
COM_InvalidParamLength =	"invalid parameter length : "
COM_InvalidParamValue =	"invalid parameter value : "
COM_InvalidAddress =	"invalid address : "
COM_InvalidBufferSize =	"invalid buffer size : "

6.4.2 RadioMessage : send a radio frame, in 'Message' mode

This function allows to send a radio frame to a known distant equipment, without waiting for an answer (see WaveCard user manual).



Currently, in relaying mode, the relay route is not automatically sent to the distant equipment.
Consequently, the user must ensure that the relay route is known by the transmitter and the receiver of a relayed frame.

VB6

```
RadioMessage (      (in) address as String,
                    (in) frame as String,
                    (in) repeater as String,
                    (out) error as String ) as String
```

C# .Net

```
string RadioMessage (string address, string frame, string repeater, string error)
```

➤ Parameters :

address	: radio address of the recipient (6 bytes)
frame	: data frame to transmit (max size : see remark below)
repeater	: list of the repeaters necessary to reach the remote module. <i>format : string containing the number of repeaters to be used, and the addresses of the repeaters consecutively (without separator)</i> <i>example : rep = "0301160430301D011604301D7D011604301D7E"</i> <i>'Nb rep Module 1 address Module 2 address Module 3 address'</i>



Remark : maximum size definition

- Point to Point mode : **Max** = 152 bytes of data
- Relaying mode : **Max** = **152** - (2 + 6 × Number of repeaters)

=> 1 repeater : 144 bytes of data,
=> 2 repeaters : 138 bytes of data,
=> 3 repeaters : 132 bytes of data.

➤ Returned values :

strResult	: echo of the sent frame
error	: = '0' if function succeeded; if not, returns a string containing an error message.



Example of 'error' values :

XML_GetInvalid =	"invalid Get method in the XML configuration file : "
XML_SetInvalid =	"invalid Set method in the XML configuration file : "
XML_ParamInvalid =	"invalid parameter in the XML configuration file : "
XML_ParamLengthInvalid =	"invalid parameter length in the XML configuration file : "
XML_GetUndefined =	"Get method not defined in the XML configuration file : "
XML_SetUndefined =	"Set method not defined in the XML configuration file : "
XML_ParamUndefined =	"parameter not defined in the XML configuration file : "
XML_ParamLengthUndefined =	"parameter length not defined in the XML configuration file : "
COM_RadioError =	"radio communication error"
COM_RS232Error =	"rs232 communication error"
COM_InvalidParamLength =	"invalid parameter length : "
COM_InvalidParamValue =	"invalid parameter value : "
COM_InvalidAddress =	"invalid address : "
COM_InvalidBufferSize =	"invalid buffer size : "

6.4.3 RadioPacket : transmit a frame with waiting of a response in 'multi-frame' mode.

this function allows to transmit a radio frame to a known distant equipment, with waiting of an answer transmitted in 'multi-frame' mode.



Currently, the Wavenis protocol does not support the multi-frame in relaying mode, it will be implemented in future developments.

Consequently, to indicate that no repeater is used, the 'repeater' field must be initialized with null string (repeater = "")

VB6

```
RadioPacket ( (in) address as String,
              (in) frame as String,
              (in) repeater as String,
              (in) timeout as int,
              (out) error as String ) as String
```

C# .Net

```
string RadioPacket (string address, string frame, string repeater, int timeout, string error)
```

➤ Parameters :

address frame repeater timeout	: radio address of the recipient (6 bytes) : data frame to transmit (max size : see remark below) : list of the repeaters necessary to reach the remote module. <i>format : string containing the number of repeaters to be used, and the addresses of the repeaters consecutively (without separator)</i> <i>example : rep = "0301160430301D011604301D7D011604301D7E"</i> <i>'Nb rep Module 1 address Module 2 address Module 3 address '</i>
---	---



Remark : maximum size definition

- Point to Point mode : **Max** = 152 bytes of data
- Relaying mode : **Max** = 152 - (2 + 6 x Number of repeaters)

=> 1 repeater : 144 bytes of data,
=> 2 repeaters : 138 bytes of data,
=> 3 repeaters : 132 bytes of data.

➤ Returned values :

strResult error	: string containing the response from the remote module. : = '0' if function succeeded; if not, returns a string containing an error message.
----------------------------------	--



Example of 'error' values :

XML_GetInvalid =	"invalid Get method in the XML configuration file : "
XML_SetInvalid =	"invalid Set method in the XML configuration file : "
XML_ParamInvalid =	"invalid parameter in the XML configuration file : "
XML_ParamLengthInvalid =	"invalid parameter length in the XML configuration file : "
XML_GetUndefined =	"Get method not defined in the XML configuration file : "
XML_SetUndefined =	"Set method not defined in the XML configuration file : "
XML_ParamUndefined =	"parameter not defined in the XML configuration file : "
XML_ParamLengthUndefined =	"parameter length not defined in the XML configuration file : "
COM_RadioError =	"radio communication error"
COM_RS232Error =	"rs232 communication error"
COM_InvalidParamLength =	"invalid parameter length : "
COM_InvalidParamValue =	"invalid parameter value : "
COM_InvalidAddress =	"invalid address : "
COM_InvalidBufferSize =	"invalid buffer size : "

6.4.4 RadioExchangePolling : send a radio frame using the 'polling' mode

This function send a radio frame to a group of known recipients, whose radio addresses were configured beforehand (via *SetPollingTable* and *GroupNumber* functions).

In addition, time separating two consecutive emissions might also be configured beforehand it, via the *SetPollingTime* method.



Remark : the configuration parameters of the Polling mode must be configured before the use of this mode of transmission.

VB6

```
RadioExchangePolling ( (in) frame as String,  
                         (out) error as String ) as String
```

C# .Net

```
string RadioExchangePolling (string frame, string error)
```

➤ Parameters :

frame : string containing the data to transmit (152 bytes max.)

➤ Return values :

strResult : returns a concatenation of responses from interrogated modules. the responses are separated with a SPACE character.



Example of 'strResult' value after a polling request on 3 modules:

"Data_module_1 Data_Module_2 Data_Module_3"

error : = '0' if function succeeded; if not, returns a string containing an error message.



Example of 'error' values :

<i>XML_GetInvalid =</i>	<i>"invalid Get method in the XML configuration file : "</i>
<i>XML_SetInvalid =</i>	<i>"invalid Set method in the XML configuration file : "</i>
<i>XML_ParamInvalid =</i>	<i>"invalid parameter in the XML configuration file : "</i>
<i>XML_ParamLengthInvalid =</i>	<i>"invalid parameter length in the XML configuration file : "</i>
<i>XML_GetUndefined =</i>	<i>"Get method not defined in the XML configuration file : "</i>
<i>XML_SetUndefined =</i>	<i>"Set method not defined in the XML configuration file : "</i>
<i>XML_ParamUndefined =</i>	<i>"parameter not defined in the XML configuration file : "</i>
<i>XML_ParamLengthUndefined =</i>	<i>"parameter length not defined in the XML configuration file : "</i>
<i>COM_RadioError =</i>	<i>"radio communication error"</i>
<i>COM_RS232Error =</i>	<i>"rs232 communication error"</i>
<i>COM_InvalidParamLength =</i>	<i>"invalid parameter length : "</i>
<i>COM_InvalidParamValue =</i>	<i>"invalid parameter value : "</i>
<i>COM_InvalidAddress =</i>	<i>"invalid address : "</i>
<i>COM_InvalidBufferSize =</i>	<i>"invalid buffer size : "</i>

APPENDIX 1 – VB6 EXAMPLE CODE

```
1 Dim WPort As IWavePort
2
3 Private Sub Form_Load()
4     ' Create the WPort instance
5     Set WPort = New Wavenis_Waveport.WavePort
6     ' Add Item Com Port number
7     selCom.AddItem ("COM1")
8     selCom.AddItem ("COM2")
9     selCom.AddItem ("COM3")
10    selCom.AddItem ("COM4")
11    selCom.AddItem ("COM5")
12    selCom.AddItem ("COM6")
13    selCom.AddItem ("COM7")
14    selCom.AddItem ("COM8")
15    selCom.AddItem ("COM9")
16    selCom.Text = selCom.List(0)
17 End Sub
18
19 Private Sub Form_Unload(Cancel As Integer)
20     If IsNull(WPort) Then
21         Else
22             WPort.Close
23         End If
24     End Sub
25
26 Private Sub open_Click()
27     Dim iCom As Integer
28     iCom = selCom.ListIndex + 1
29     'Select the Com Port number
30     WPort.SetCommPortNumber (iCom)
31     Dim bSuccess As Boolean
32     'Initialize the WPort
33     bSuccess = WPort.Init(9600, 8, 0, 1, 0)
34     If (bSuccess) Then
35         'If succeed, enable the alarm event handler timer
36         received.Caption = "OK"
37         Timer1.Enabled = True   Else
38         received.Caption = "NOK"
39         Timer1.Enabled = False
40     End If
41 End Sub
42
43 Private Sub sendFrame_Click()
44     Dim strError As String
45     Dim strResult As String
46     ' Send the frame and wait for the response
47     strResult = WPort.RadioExchange(address.Text, data.Text, "", 6100, strError)
```

```
48 If strResult = 0 Then
49     received.Caption = strError
50 Else
51     received.Caption = strResult
52 End If
53 End Sub
54
55 Private Sub sendMessage Click()
56     Dim strError As String
57     Dim strResult As String
58     ' Send the Message
59     strResult = WPort.RadioMessage(address.Text, data.Text, "", strError)
60     received.Caption = result
61 End Sub
62
63 Private Sub Timer1 Timer()
64     Dim WEvent As Wavenis Waveport.IWavenisEvent
65     'Received the event if one is existing
66     If (WPort.WatchLink(WEvent)) Then
67         received.Caption = WEvent.GetAddress + "=>" + WEvent.GetValue
68     End If
69 End Sub
```

APPENDIX 2 – VC++6 EXAMPLE CODE

```

1 // WPortCpp6Dlg.h : header file
2 //
3
4 #if !defined(AFX_WPORTCPP6DLG_H__3889C184_73FB_44B0_BF88_D234786E5532__INCLUDED_)
5 #define AFX_WPORTCPP6DLG_H__3889C184_73FB_44B0_BF88_D234786E5532__INCLUDED_
6
7 #if _MSC_VER > 1000
8 #pragma once
9#endif // _MSC_VER > 1000
10
11 // Add Wavenis.Waveport.tlb reference
12 #import "c:\windows\system32\coronis-systems\win32\Wavenis.Waveport.tlb" raw_interfaces_only
13
14 ///////////////////////////////////////////////////////////////////
15 // CWPortCpp6Dlg dialog
16
17 class CWPortCpp6Dlg : public CDialog
18 {
19 // Construction
20 public:
21     bool InitWavePort(HRESULT hr);
22     CWPortCpp6Dlg(CWnd* pParent = NULL); // standard constructor
23
24 // Dialog Data
25     //{{AFX_DATA(CWPortCpp6Dlg)
26     enum { IDD = IDD_WPORTCPP6_DIALOG };
27     CComboBox m_selCom;
28     CStatic m_received;
29     CEdit m_data;
30     CEdit m_address;
31 //}}AFX_DATA
32
33 // ClassWizard generated virtual function overrides
34 //{{AFX_VIRTUAL(CWPortCpp6Dlg)
35 protected:
36     virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
37 //}}AFX_VIRTUAL
38
39 // Implementation
40     HRESULT hr;
41     UINT m_timer1;
42     Wavenis_Waveport::IWavePortPtr Wport;
43
44 protected:

```

```

45     HICON m_hIcon;
46
47     // Generated message map functions
48     //{{AFX_MSG(CWPortCpp6Dlg)
49     virtual BOOL OnInitDialog();
50     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
51     afx_msg void OnPaint();
52     afx_msg HCURSOR OnQueryDragIcon();
53     afx_msg void OnOpen();
54     afx_msg void OnSendFrame();
55     afx_msg void OnSendMessage();
56     afx_msg void OnTimer(UINT nIDEvent);
57     //}}AFX_MSG
58     DECLARE_MESSAGE_MAP()
59 };
60
61 //{{AFX_INSERT_LOCATION}
62 // Microsoft Visual C++ will insert additional declarations immediately before the previous line.
63
64 #endif // !defined(AFX_WPORTCPP6DLG_H__3889C184_73FB_44B0_BF88_D234786E5532_INCLUDED_)
65
66 //////////////////////////////////////////////////////////////////
67 //////////////////////////////////////////////////////////////////
68 //////////////////////////////////////////////////////////////////
69
70 // WPortCpp6Dlg.cpp : implementation file
71 //
72 #include "stdafx.h"
73 #include "WPortCpp6.h"
74 #include "WPortCpp6Dlg.h"
75
76 #ifdef _DEBUG
77 #define new DEBUG_NEW
78 #undef THIS_FILE
79 static char THIS_FILE[] = __FILE__;
80 #endif
81
82 //////////////////////////////////////////////////////////////////
83 // CAboutDlg dialog used for App About
84
85 class CAboutDlg : public CDialog
86 {
87 public:
88     CAboutDlg();
89
90 // Dialog Data

```

```
91 //{{AFX_DATA(CAboutDlg)
92 enum { IDD = IDD_ABOUTBOX };
93 //}}AFX_DATA
94
95 // ClassWizard generated virtual function overrides
96 //{{AFX_VIRTUAL(CAboutDlg)
97 protected:
98 virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
99 //}}AFX_VIRTUAL
100
101 // Implementation
102 protected:
103 //{{AFX_MSG(CAboutDlg)
104 //}}AFX_MSG
105 DECLARE_MESSAGE_MAP()
106 };
107
108 CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
109 {
110 //{{AFX_DATA_INIT(CAboutDlg)
111 //}}AFX_DATA_INIT
112 }
113
114 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
115 {
116     CDialog::DoDataExchange(pDX);
117 //{{AFX_DATA_MAP(CAboutDlg)
118 //}}AFX_DATA_MAP
119 }
120
121 BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
122 //{{AFX_MSG_MAP(CAboutDlg)
123 // No message handlers
124 //}}AFX_MSG_MAP
125 END_MESSAGE_MAP()
126
127 /////////////////////////////////
128 // CWPortCpp6Dlg dialog
129
130 CWPortCpp6Dlg::CWPortCpp6Dlg(CWnd* pParent /*=NULL*/)
131     : CDialog(CWPortCpp6Dlg::IDD, pParent)
132 {
133 //{{AFX_DATA_INIT(CWPortCpp6Dlg)
134 // NOTE: the ClassWizard will add member initialization here
135 //}}AFX_DATA_INIT
136 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
```

```
137     m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
138
139     // Initialization of the timer
140     m_timer1 = 1;
141 }
142
143 void CWPortCpp6Dlg::DoDataExchange(CDataExchange* pDX)
144 {
145     CDialog::DoDataExchange(pDX);
146     //{{AFX_DATA_MAP(CWPortCpp6Dlg)
147     DDX_Control(pDX, IDC_SEL_COM, m_selCom);
148     DDX_Control(pDX, IDC_RECEIVED, m_received);
149     DDX_Control(pDX, IDC_DATA, m_data);
150     DDX_Control(pDX, IDC_ADDRESS, m_address);
151     //}}AFX_DATA_MAP
152 }
153
154 BEGIN_MESSAGE_MAP(CWPortCpp6Dlg, CDialog)
155     //{{AFX_MSG_MAP(CWPortCpp6Dlg)
156     ON_WM_SYSCOMMAND()
157     ON_WM_PAINT()
158     ON_WM_QUERYDRAGICON()
159     ON_BN_CLICKED(IDC_OPEN, OnOpen)
160     ON_BN_CLICKED(IDC_SEND_FRAME, OnSendFrame)
161     ON_BN_CLICKED(IDC_SEND_MESSAGE, OnSendMessage)
162     ON_WM_TIMER()
163     //}}AFX_MSG_MAP
164 END_MESSAGE_MAP()
165
166 //////////////////////////////////////////////////////////////////
167 // CWPortCpp6Dlg message handlers
168
169 BOOL CWPortCpp6Dlg::OnInitDialog()
170 {
171     CDialog::OnInitDialog();
172
173     // Add "About..." menu item to system menu.
174
175     // IDM_ABOUTBOX must be in the system command range.
176     ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
177     ASSERT(IDM_ABOUTBOX < 0xF000);
178
179     CMenu* pSysMenu = GetSystemMenu(FALSE);
180     if (pSysMenu != NULL)
181     {
182         CString strAboutMenu;
```

```
183     strAboutMenu.LoadString(IDS_ABOUTBOX);
184     if (!strAboutMenu.IsEmpty())
185     {
186         pSysMenu->AppendMenu(MF_SEPARATOR);
187         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
188     }
189 }
190
191 // Set the icon for this dialog. The framework does this automatically
192 // when the application's main window is not a dialog
193 SetIcon(m_hIcon, TRUE);           // Set big icon
194 SetIcon(m_hIcon, FALSE);         // Set small icon
195
196 // TODO: Add extra initialization here
197 hr = NULL;
198 bool bStatus = InitWavePort(hr);
199
200 m_selCom.AddString("COM1");
201 m_selCom.AddString("COM2");
202 m_selCom.AddString("COM3");
203 m_selCom.AddString("COM4");
204 m_selCom.AddString("COM5");
205 m_selCom.AddString("COM6");
206 m_selCom.AddString("COM7");
207 m_selCom.AddString("COM8");
208 m_selCom.AddString("COM9");
209 m_selCom.SetCurSel(0);
210 m_address.SetWindowText("011603300373");
211 m_data.SetWindowText("01");
212
213 return TRUE; // return TRUE unless you set the focus to a control
214 }
215
216 void CWPortCpp6Dlg::OnSysCommand(UINT nID, LPARAM lParam)
217 {
218     if ((nID & 0xFFFF) == IDM_ABOUTBOX)
219     {
220         CAaboutDlg dlgAbout;
221         dlgAbout.DoModal();
222     }
223     else
224     {
225         CDialog::OnSysCommand(nID, lParam);
226     }
227 }
228 }
```

```
229 // If you add a minimize button to your dialog, you will need the code below
230 // to draw the icon. For MFC applications using the document/view model,
231 // this is automatically done for you by the framework.
232
233 void CWPortCpp6Dlg::OnPaint()
234 {
235     if (IsIconic())
236     {
237         CPaintDC dc(this); // device context for painting
238
239         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
240
241         // Center icon in client rectangle
242         int cxIcon = GetSystemMetrics(SM_CXICON);
243         int cyIcon = GetSystemMetrics(SM_CYICON);
244         CRect rect;
245         GetClientRect(&rect);
246         int x = (rect.Width() - cxIcon + 1) / 2;
247         int y = (rect.Height() - cyIcon + 1) / 2;
248
249         // Draw the icon
250         dc.DrawIcon(x, y, m_hIcon);
251     }
252     else
253     {
254         CDialog::OnPaint();
255     }
256 }
257
258 // The system calls this to obtain the cursor to display while the user drags
259 // the minimized window.
260 HCURSOR CWPortCpp6Dlg::OnQueryDragIcon()
261 {
262     return (HCURSOR) m_hIcon;
263 }
264
265 void CWPortCpp6Dlg::OnOpen()
266 {
267     // TODO: Add your control notification handler code here
268
269     //Variables
270     short iCom=0;
271     VARIANT_BOOL bSuccess = VARIANT_FALSE;
272
273     iCom = m_selCom.GetCurSel();
274     iCom++;
```

```
275  
276 // Select the Com Port  
277 hr = WPort->SetCommPortNumber(iCom);  
278  
279 // Init the WPort  
280 hr = WPort->Init(9600,8,0,1,0,&bSuccess);  
281  
282 if (SUCCEEDED(hr) && bSuccess)  
283 {  
284     m_received.SetWindowText("OK");  
285     // Setup the timer for the Event Handler  
286     SetTimer(m_timer1,230,NULL);  
287 }  
288 else  
289 {  
290     m_received.SetWindowText("NOK");  
291 }  
292 }  
293  
294 void CWPortCpp6Dlg::OnSendFrame()  
295 {  
296     // TODO: Add your control notification handler code here  
297  
298     CComBSTR bzAddress=NULL;  
299     CComBSTR bzFrame=NULL;  
300     CComBSTR bzRepeater=NULL;  
301     CComBSTR bzError=NULL;  
302     CComBSTR bzResult=NULL;  
303  
304     CString strAddress;  
305     CString strFrame;  
306     CString strResult;  
307  
308     m_address.GetWindowText(strAddress);  
309     m_data.GetWindowText(strFrame);  
310  
311     bzAddress.Append(strAddress);  
312     bzFrame.Append(strFrame);  
313     bzRepeater.Append("");  
314  
315     hr = WPort->RadioExchange(bzAddress,bzFrame,bzRepeater,6100,&bzError,&bzResult);  
316  
317     strResult = bzAddress.Detach();  
318     strResult += bzFrame.Detach();  
319     strResult += bzError.Detach();  
320     strResult += bzResult.Detach();
```

```
321
322     m_received.SetWindowText(strResult);
323
324 }
325
326 void CWPortCpp6Dlg::OnSendMessage()
327 {
328     // TODO: Add your control notification handler code here
329
330     CComBSTR bzAddress=NULL;
331     CComBSTR bzFrame=NULL;
332     CComBSTR bzRepeater=NULL;
333     CComBSTR bzError=NULL;
334     CComBSTR bzResult=NULL;
335
336     CString strAddress;
337     CString strFrame;
338     CString strResult;
339
340     m_address.GetWindowText(strAddress);
341     m_data.GetWindowText(strFrame);
342
343     bzAddress.Append(strAddress);
344     bzFrame.Append(strFrame);
345     bzRepeater.Append("");
346
347     hr = WPort->RadioMessage(bzAddress,bzFrame,bzRepeater,&bzError,&bzResult);
348
349     strResult = bzAddress.Detach();
350     strResult += bzFrame.Detach();
351     strResult += bzError.Detach();
352     strResult += bzResult.Detach();
353
354     m_received.SetWindowText(strResult);
355 }
356
357 bool CWPortCpp6Dlg::InitWavePort(HRESULT hr)
358 {
359     CoInitialize(NULL);
360
361     CLSID CLSID_WavePort = CLSID_NULL;
362     IID IID_IWavePort = IID_NULL;
363
364     // class WAVEPORT
365     CLSIDFromString(L"{C02F4CD9-A172-48B3-9878-EAA177ED7FE7}",&CLSID_WavePort);
366 }
```

```

367     // interface IWavePort
368     IIDFromString(L"{BE5B7BD8-ABA8-49ED-A2DA-9876B6239BED}",&IID_IWAVEPORT);
369
370     hr = ::CoCreateInstance(CLSID_WavePort,
371                             NULL,
372                             CLSCTX_INPROC_SERVER,
373                             IID_IWAVEPORT,
374                             reinterpret_cast<void**>(&WPort));
375
376     if (SUCCEEDED(hr))
377     {
378         return true;
379     }
380     else
381     {
382         return false;
383     }
384 }
385
386 void CWPortCpp6Dlg::OnTimer(UINT nIDEvent)
387 {
388     // TODO: Add your message handler code here and/or call default
389     If (m_timer1 == nIDEvent)
390     {
391         Wavenis_Waveport::IWavenisEventPtr WEvent=NULL;
392         VARIANT_BOOL result=VARIANT_FALSE;
393         HRESULT hr = WPort->WatchLink(&WEvent,&result);
394         if (VARIANT_TRUE==result)
395         {
396             CComBSTR adr;
397
398             CComBSTR data;
399             CString strResult;
400             hr=WEvent->GetAddress(&adr);
401             hr=WEvent->GetValue(&data);
402             strResult = CString(adr.Detach()) + "=" + data.Detach();
403
404             m_received.SetWindowText(strResult);
405
406         }
407     }
408 }
409 CDialog::OnTimer(nIDEvent);
410 }
411
412

```

APPENDIX 3 – VB.NET EXAMPLE CODE

```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3     Dim WPort As Wavenis.WavePort
4
5 #Region " Code généré par le Concepteur Windows Form "
6
7     Public Sub New()
8         MyBase.New()
9
10        'Cet appel est requis par le Concepteur Windows Form.
11        InitializeComponent()
12
13        'Ajoutez une initialisation quelconque après l'appel InitializeComponent()
14        WPort = New Wavenis.WavePort
15
16        selCom.Items.Add("COM1")
17        selCom.Items.Add("COM2")
18        selCom.Items.Add("COM3")
19        selCom.Items.Add("COM4")
20        selCom.Items.Add("COM5")
21        selCom.Items.Add("COM6")
22        selCom.Items.Add("COM7")
23        selCom.Items.Add("COM8")
24        selCom.Items.Add("COM9")
25        selCom.SelectedIndex = 0
26    End Sub
27
28    Private Sub alarmRang(ByVal sender As Object, ByVal WEvent As Wavenis.IWavenisEvent)
29        received.Text = WEvent.GetAddress() + "=" + WEvent.GetValue()
30    End Sub
31
32    'La méthode substituée Dispose du formulaire pour nettoyer la liste des composants.
33    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
34        If disposing Then
35            If Not (components Is Nothing) Then
36                components.Dispose()
37            End If
38        End If
39        If Not (WPort Is Nothing) Then
40            WPort.Close()
41        End If
42        MyBase.Dispose(disposing)
43    End Sub
44
45    'Requis par le Concepteur Windows Form
46    Private components As System.ComponentModel.IContainer
47
48
```

```
49 'REMARQUE : la procédure suivante est requise par le Concepteur Windows Form
50 'Elle peut être modifiée en utilisant le Concepteur Windows Form.
51 'Ne la modifiez pas en utilisant l'éditeur de code.
52 Friend WithEvents data As System.Windows.Forms.TextBox
53 Friend WithEvents received As System.Windows.Forms.Label
54 Friend WithEvents address As System.Windows.Forms.TextBox
55 Friend WithEvents sendFrame As System.Windows.Forms.Button
56 Friend WithEvents sendMessage As System.Windows.Forms.Button
57 Friend WithEvents open As System.Windows.Forms.Button
58 Friend WithEvents selCom As System.Windows.Forms.ComboBox
59
60 Private Sub InitializeComponent()
61     Me.data = New System.Windows.Forms.TextBox
62     Me.received = New System.Windows.Forms.Label
63     Me.address = New System.Windows.Forms.TextBox
64     Me.sendFrame = New System.Windows.Forms.Button
65     Me.sendMessage = New System.Windows.Forms.Button
66     Me.open = New System.Windows.Forms.Button
67     Me.selCom = New System.Windows.Forms.ComboBox
68     Me.SuspendLayout()
69
70     'data
71
72     Me.data.Location = New System.Drawing.Point(18, 76)
73     Me.data.Name = "data"
74     Me.data.Size = New System.Drawing.Size(256, 20)
75     Me.data.TabIndex = 13
76     Me.data.Text = "01"
77
78     'received
79
80     Me.received.Location = New System.Drawing.Point(18, 108)
81     Me.received.Name = "received"
82     Me.received.Size = New System.Drawing.Size(256, 152)
83     Me.received.TabIndex = 12
84
85     'address
86
87     Me.address.Location = New System.Drawing.Point(18, 44)
88     Me.address.Name = "address"
89     Me.address.Size = New System.Drawing.Size(88, 20)
90     Me.address.TabIndex = 11
91     Me.address.Text = "011603300373"
92
93     'sendFrame
94
95     Me.sendFrame.Location = New System.Drawing.Point(114, 44)
96     Me.sendFrame.Name = "sendFrame"
97     Me.sendFrame.Size = New System.Drawing.Size(72, 24)
98     Me.sendFrame.TabIndex = 10
```

```
99 Me.sendFrame.Text = "SendFrame"
'
100
101 'sendMessage
'
102
103 Me.sendMessage.Location = New System.Drawing.Point(186, 44)
104 Me.sendMessage.Name = "sendMessage"
105 Me.sendMessage.Size = New System.Drawing.Size(88, 24)
106 Me.sendMessage.TabIndex = 9
107 Me.sendMessage.Text = "SendMessage"
'
108
109 'open
'
110
111 Me.open.Location = New System.Drawing.Point(114, 12)
112 Me.open.Name = "open"
113 Me.open.Size = New System.Drawing.Size(40, 24)
114 Me.open.TabIndex = 8
115 Me.open.Text = "Open"
'
116
117 'selCom
'
118
119 Me.selCom.Location = New System.Drawing.Point(18, 12)
120 Me.selCom.Name = "selCom"
121 Me.selCom.Size = New System.Drawing.Size(64, 21)
122 Me.selCom.TabIndex = 7
'
123
124 'Form1
'
125
126 Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
127 Me.ClientSize = New System.Drawing.Size(292, 273)
128 Me.Controls.Add(Me.data)
129 Me.Controls.Add(Me.received)
130 Me.Controls.Add(Me.address)
131 Me.Controls.Add(Me.sendFrame)
132 Me.Controls.Add(Me.sendMessage)
133 Me.Controls.Add(Me.open)
134 Me.Controls.Add(Me.selCom)
135 Me.Name = "Form1"
136 Me.Text = "Form1"
137 Me.ResumeLayout(False)
'
138
139 End Sub
'
140
141 #End Region
'
142
143
144 Private Sub open_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles open.Click
145     'Com Port selection
146     Dim iCom As Integer
147     iCom = selCom.SelectedIndex
148     iCom = iCom + 1
```

```
149 WPort.SetCommPortNumber(iCom)
150
151 'Initialization of the WavePort
152 If (WPort.Init(9600, 8, 0, 1, 0)) Then
153     received.Text = "OK"
154     ' If result successed, set up the alarm event handler
155     AddHandler WPort.Alarm, AddressOf Me.alarmRang
156 Else
157     received.Text = "NOK"
158 End If
159 End Sub
160
161 Private Sub sendFrame_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles sendFrame.Click
162     Dim strError As String
163     Dim strResult As String
164     Dim repeater As String
165     ' Send a frame and wait for the response
166     strResult = WPort.RadioExchange(address.Text, data.Text, repeater, 6100, strError)
167     If (strResult = "") Then
168         received.Text = strError
169     Else
170         received.Text = strResult
171     End If
172 End Sub
173
174 Private Sub sendMessage_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles sendMessage.Click
175     Dim strError As String
176     Dim strResult As String
177     Dim repeater As String
178     ' Send a frame
179     strResult = WPort.RadioMessage(address.Text, data.Text, repeater, strError)
180     If (strResult = "") Then
181         received.Text = strResult
182     Else
183         received.Text = strError
184     End If
185 End Sub
186 End Class
```

APPENDIX 4 – C# .NET EXAMPLE CODE FOR WIN32

```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7  using Wavenis;
8  namespace WPortCSharp.Net
9  {
10     /// <summary>
11     /// Description résumée de Form1.
12     /// </summary>
13     public class Form1 : System.Windows.Forms.Form
14     {
15         private System.Windows.Forms.ComboBox selCom;
16         private System.Windows.Forms.Button open;
17         private System.Windows.Forms.Button sendMessage;
18         private System.Windows.Forms.Button sendFrame;
19         private System.Windows.Forms.TextBox address;
20         private System.Windows.Forms.TextBox data;
21         private Wavenis.WavePort WPort;
22         private System.Windows.Forms.Label received;
23         /// <summary>
24         /// Variable nécessaire au concepteur.
25         /// </summary>
26         private System.ComponentModel.Container components = null;
27
28         public Form1()
29         {
30             //
31             // Requis pour la prise en charge du Concepteur Windows Forms
32             //
33             InitializeComponent();
34
35             //
36             // TODO : ajoutez le code du constructeur après l'appel à InitializeComponent
37             //
38
39             WPort = new WavePort();
40
41             string[] comList = new string[9];
42             for (int nIndex=1; nIndex<10; nIndex++)
43             {
44                 string text= "COM" + nIndex.ToString();
45                 comList[nIndex-1]=text;
46             }
47         }
48
49         protected override void Dispose(bool disposing)
50         {
51             if (disposing)
52             {
53                 if (components != null)
54                     components.Dispose();
55             }
56             base.Dispose(disposing);
57         }
58
59         #region Windows Form Designer generated code
60
61         /// <summary>
62         /// Généré par le Concepteur Windows Forms
63         /// Pour modifier ce champ, modifiez le contenu du code
64         /// dans le Concepteur Windows Forms.
65         /// </summary>
66         [System.Diagnostics.DebuggerStepThrough()]
67         private System.ComponentModel.IContainer components;
68
69         private System.Windows.Forms.Button button1;
70         private System.Windows.Forms.Label label1;
71         private System.Windows.Forms.TextBox textBox1;
72         private System.Windows.Forms.ComboBox comboBox1;
73         private System.Windows.Forms.Button button2;
74         private System.Windows.Forms.Button button3;
75         private System.Windows.Forms.Button button4;
76         private System.Windows.Forms.Button button5;
77         private System.Windows.Forms.Button button6;
78         private System.Windows.Forms.Button button7;
79         private System.Windows.Forms.Button button8;
80         private System.Windows.Forms.Button button9;
81         private System.Windows.Forms.Button button10;
82
83         #endregion
84
85         private void InitializeComponent()
86         {
87             this.components = new System.ComponentModel.Container();
88             this.button1 = new System.Windows.Forms.Button();
89             this.label1 = new System.Windows.Forms.Label();
90             this.textBox1 = new System.Windows.Forms.TextBox();
91             this.comboBox1 = new System.Windows.Forms.ComboBox();
92             this.button2 = new System.Windows.Forms.Button();
93             this.button3 = new System.Windows.Forms.Button();
94             this.button4 = new System.Windows.Forms.Button();
95             this.button5 = new System.Windows.Forms.Button();
96             this.button6 = new System.Windows.Forms.Button();
97             this.button7 = new System.Windows.Forms.Button();
98             this.button8 = new System.Windows.Forms.Button();
99             this.button9 = new System.Windows.Forms.Button();
100            this.button10 = new System.Windows.Forms.Button();
101            this.received = new System.Windows.Forms.Label();
102            this.SuspendLayout();
103
104            this.button1.Location = new System.Drawing.Point(12, 12);
105            this.button1.Name = "button1";
106            this.button1.Size = new System.Drawing.Size(75, 23);
107            this.button1.TabIndex = 0;
108            this.button1.Text = "button1";
109            this.button1.UseVisualStyleBackColor = true;
110            this.button1.Click += new System.EventHandler(this.button1_Click);
111
112            this.label1.AutoSize = true;
113            this.label1.Location = new System.Drawing.Point(12, 37);
114            this.label1.Name = "label1";
115            this.label1.Size = new System.Drawing.Size(57, 16);
116            this.label1.TabIndex = 1;
117            this.label1.Text = "label1";
118
119            this.textBox1.Location = new System.Drawing.Point(12, 62);
120            this.textBox1.Name = "textBox1";
121            this.textBox1.Size = new System.Drawing.Size(100, 20);
122            this.textBox1.TabIndex = 2;
123
124            this.comboBox1.FormattingEnabled = true;
125            this.comboBox1.Location = new System.Drawing.Point(12, 87);
126            this.comboBox1.Name = "comboBox1";
127            this.comboBox1.Size = new System.Drawing.Size(121, 21);
128            this.comboBox1.TabIndex = 3;
129
130            this.button2.Location = new System.Drawing.Point(12, 112);
131            this.button2.Name = "button2";
132            this.button2.Size = new System.Drawing.Size(75, 23);
133            this.button2.TabIndex = 4;
134            this.button2.Text = "button2";
135            this.button2.UseVisualStyleBackColor = true;
136
137            this.button3.Location = new System.Drawing.Point(87, 112);
138            this.button3.Name = "button3";
139            this.button3.Size = new System.Drawing.Size(75, 23);
140            this.button3.TabIndex = 5;
141            this.button3.Text = "button3";
142            this.button3.UseVisualStyleBackColor = true;
143
144            this.button4.Location = new System.Drawing.Point(162, 112);
145            this.button4.Name = "button4";
146            this.button4.Size = new System.Drawing.Size(75, 23);
147            this.button4.TabIndex = 6;
148            this.button4.Text = "button4";
149            this.button4.UseVisualStyleBackColor = true;
150
151            this.button5.Location = new System.Drawing.Point(237, 112);
152            this.button5.Name = "button5";
153            this.button5.Size = new System.Drawing.Size(75, 23);
154            this.button5.TabIndex = 7;
155            this.button5.Text = "button5";
156            this.button5.UseVisualStyleBackColor = true;
157
158            this.button6.Location = new System.Drawing.Point(312, 112);
159            this.button6.Name = "button6";
160            this.button6.Size = new System.Drawing.Size(75, 23);
161            this.button6.TabIndex = 8;
162            this.button6.Text = "button6";
163            this.button6.UseVisualStyleBackColor = true;
164
165            this.button7.Location = new System.Drawing.Point(387, 112);
166            this.button7.Name = "button7";
167            this.button7.Size = new System.Drawing.Size(75, 23);
168            this.button7.TabIndex = 9;
169            this.button7.Text = "button7";
170            this.button7.UseVisualStyleBackColor = true;
171
172            this.button8.Location = new System.Drawing.Point(462, 112);
173            this.button8.Name = "button8";
174            this.button8.Size = new System.Drawing.Size(75, 23);
175            this.button8.TabIndex = 10;
176            this.button8.Text = "button8";
177            this.button8.UseVisualStyleBackColor = true;
178
179            this.button9.Location = new System.Drawing.Point(537, 112);
180            this.button9.Name = "button9";
181            this.button9.Size = new System.Drawing.Size(75, 23);
182            this.button9.TabIndex = 11;
183            this.button9.Text = "button9";
184            this.button9.UseVisualStyleBackColor = true;
185
186            this.button10.Location = new System.Drawing.Point(612, 112);
187            this.button10.Name = "button10";
188            this.button10.Size = new System.Drawing.Size(75, 23);
189            this.button10.TabIndex = 12;
190            this.button10.Text = "button10";
191            this.button10.UseVisualStyleBackColor = true;
192
193            this.received.Location = new System.Drawing.Point(12, 137);
194            this.received.Name = "received";
195            this.received.Size = new System.Drawing.Size(100, 20);
196            this.received.TabIndex = 13;
197
198            this.ResumeLayout(false);
199        }
200
201        private void button1_Click(object sender, EventArgs e)
202        {
203            WPort.open();
204        }
205
206        private void button2_Click(object sender, EventArgs e)
207        {
208            WPort.sendMessage();
209        }
210
211        private void button3_Click(object sender, EventArgs e)
212        {
213            WPort.sendFrame();
214        }
215
216        private void button4_Click(object sender, EventArgs e)
217        {
218            WPort.selCom();
219        }
220
221        private void button5_Click(object sender, EventArgs e)
222        {
223            WPort.address();
224        }
225
226        private void button6_Click(object sender, EventArgs e)
227        {
228            WPort.data();
229        }
230
231        private void button7_Click(object sender, EventArgs e)
232        {
233            WPort.close();
234        }
235
236        private void button8_Click(object sender, EventArgs e)
237        {
238            WPort.received();
239        }
240
241        private void button9_Click(object sender, EventArgs e)
242        {
243            WPort.frame();
244        }
245
246        private void button10_Click(object sender, EventArgs e)
247        {
248            WPort.comList();
249        }
250
251        private void WPort_open(object sender, EventArgs e)
252        {
253            WPort.open();
254        }
255
256        private void WPort_close(object sender, EventArgs e)
257        {
258            WPort.close();
259        }
260
261        private void WPort_sendMessage(object sender, EventArgs e)
262        {
263            WPort.sendMessage();
264        }
265
266        private void WPort_sendFrame(object sender, EventArgs e)
267        {
268            WPort.sendFrame();
269        }
270
271        private void WPort_selCom(object sender, EventArgs e)
272        {
273            WPort.selCom();
274        }
275
276        private void WPort_address(object sender, EventArgs e)
277        {
278            WPort.address();
279        }
280
281        private void WPort_data(object sender, EventArgs e)
282        {
283            WPort.data();
284        }
285
286        private void WPort_received(object sender, EventArgs e)
287        {
288            WPort.received();
289        }
290
291        private void WPort_frame(object sender, EventArgs e)
292        {
293            WPort.frame();
294        }
295
296        private void WPort_comList(object sender, EventArgs e)
297        {
298            WPort.comList();
299        }
300
301        private void WPort_close(object sender, EventArgs e)
302        {
303            WPort.close();
304        }
305
306        private void WPort_sendFrame(object sender, EventArgs e)
307        {
308            WPort.sendFrame();
309        }
310
311        private void WPort_sendMessage(object sender, EventArgs e)
312        {
313            WPort.sendMessage();
314        }
315
316        private void WPort_received(object sender, EventArgs e)
317        {
318            WPort.received();
319        }
320
321        private void WPort_address(object sender, EventArgs e)
322        {
323            WPort.address();
324        }
325
326        private void WPort_data(object sender, EventArgs e)
327        {
328            WPort.data();
329        }
330
331        private void WPort_close(object sender, EventArgs e)
332        {
333            WPort.close();
334        }
335
336        private void WPort_sendFrame(object sender, EventArgs e)
337        {
338            WPort.sendFrame();
339        }
340
341        private void WPort_sendMessage(object sender, EventArgs e)
342        {
343            WPort.sendMessage();
344        }
345
346        private void WPort_received(object sender, EventArgs e)
347        {
348            WPort.received();
349        }
350
351        private void WPort_address(object sender, EventArgs e)
352        {
353            WPort.address();
354        }
355
356        private void WPort_data(object sender, EventArgs e)
357        {
358            WPort.data();
359        }
360
361        private void WPort_close(object sender, EventArgs e)
362        {
363            WPort.close();
364        }
365
366        private void WPort_sendFrame(object sender, EventArgs e)
367        {
368            WPort.sendFrame();
369        }
370
371        private void WPort_sendMessage(object sender, EventArgs e)
372        {
373            WPort.sendMessage();
374        }
375
376        private void WPort_received(object sender, EventArgs e)
377        {
378            WPort.received();
379        }
380
381        private void WPort_address(object sender, EventArgs e)
382        {
383            WPort.address();
384        }
385
386        private void WPort_data(object sender, EventArgs e)
387        {
388            WPort.data();
389        }
390
391        private void WPort_close(object sender, EventArgs e)
392        {
393            WPort.close();
394        }
395
396        private void WPort_sendFrame(object sender, EventArgs e)
397        {
398            WPort.sendFrame();
399        }
400
401        private void WPort_sendMessage(object sender, EventArgs e)
402        {
403            WPort.sendMessage();
404        }
405
406        private void WPort_received(object sender, EventArgs e)
407        {
408            WPort.received();
409        }
410
411        private void WPort_address(object sender, EventArgs e)
412        {
413            WPort.address();
414        }
415
416        private void WPort_data(object sender, EventArgs e)
417        {
418            WPort.data();
419        }
420
421        private void WPort_close(object sender, EventArgs e)
422        {
423            WPort.close();
424        }
425
426        private void WPort_sendFrame(object sender, EventArgs e)
427        {
428            WPort.sendFrame();
429        }
430
431        private void WPort_sendMessage(object sender, EventArgs e)
432        {
433            WPort.sendMessage();
434        }
435
436        private void WPort_received(object sender, EventArgs e)
437        {
438            WPort.received();
439        }
440
441        private void WPort_address(object sender, EventArgs e)
442        {
443            WPort.address();
444        }
445
446        private void WPort_data(object sender, EventArgs e)
447        {
448            WPort.data();
449        }
450
451        private void WPort_close(object sender, EventArgs e)
452        {
453            WPort.close();
454        }
455
456        private void WPort_sendFrame(object sender, EventArgs e)
457        {
458            WPort.sendFrame();
459        }
460
461        private void WPort_sendMessage(object sender, EventArgs e)
462        {
463            WPort.sendMessage();
464        }
465
466        private void WPort_received(object sender, EventArgs e)
467        {
468            WPort.received();
469        }
470
471        private void WPort_address(object sender, EventArgs e)
472        {
473            WPort.address();
474        }
475
476        private void WPort_data(object sender, EventArgs e)
477        {
478            WPort.data();
479        }
480
481        private void WPort_close(object sender, EventArgs e)
482        {
483            WPort.close();
484        }
485
486        private void WPort_sendFrame(object sender, EventArgs e)
487        {
488            WPort.sendFrame();
489        }
490
491        private void WPort_sendMessage(object sender, EventArgs e)
492        {
493            WPort.sendMessage();
494        }
495
496        private void WPort_received(object sender, EventArgs e)
497        {
498            WPort.received();
499        }
500
501        private void WPort_address(object sender, EventArgs e)
502        {
503            WPort.address();
504        }
505
506        private void WPort_data(object sender, EventArgs e)
507        {
508            WPort.data();
509        }
510
511        private void WPort_close(object sender, EventArgs e)
512        {
513            WPort.close();
514        }
515
516        private void WPort_sendFrame(object sender, EventArgs e)
517        {
518            WPort.sendFrame();
519        }
520
521        private void WPort_sendMessage(object sender, EventArgs e)
522        {
523            WPort.sendMessage();
524        }
525
526        private void WPort_received(object sender, EventArgs e)
527        {
528            WPort.received();
529        }
530
531        private void WPort_address(object sender, EventArgs e)
532        {
533            WPort.address();
534        }
535
536        private void WPort_data(object sender, EventArgs e)
537        {
538            WPort.data();
539        }
540
541        private void WPort_close(object sender, EventArgs e)
542        {
543            WPort.close();
544        }
545
546        private void WPort_sendFrame(object sender, EventArgs e)
547        {
548            WPort.sendFrame();
549        }
550
551        private void WPort_sendMessage(object sender, EventArgs e)
552        {
553            WPort.sendMessage();
554        }
555
556        private void WPort_received(object sender, EventArgs e)
557        {
558            WPort.received();
559        }
560
561        private void WPort_address(object sender, EventArgs e)
562        {
563            WPort.address();
564        }
565
566        private void WPort_data(object sender, EventArgs e)
567        {
568            WPort.data();
569        }
570
571        private void WPort_close(object sender, EventArgs e)
572        {
573            WPort.close();
574        }
575
576        private void WPort_sendFrame(object sender, EventArgs e)
577        {
578            WPort.sendFrame();
579        }
580
581        private void WPort_sendMessage(object sender, EventArgs e)
582        {
583            WPort.sendMessage();
584        }
585
586        private void WPort_received(object sender, EventArgs e)
587        {
588            WPort.received();
589        }
590
591        private void WPort_address(object sender, EventArgs e)
592        {
593            WPort.address();
594        }
595
596        private void WPort_data(object sender, EventArgs e)
597        {
598            WPort.data();
599        }
600
601        private void WPort_close(object sender, EventArgs e)
602        {
603            WPort.close();
604        }
605
606        private void WPort_sendFrame(object sender, EventArgs e)
607        {
608            WPort.sendFrame();
609        }
610
611        private void WPort_sendMessage(object sender, EventArgs e)
612        {
613            WPort.sendMessage();
614        }
615
616        private void WPort_received(object sender, EventArgs e)
617        {
618            WPort.received();
619        }
620
621        private void WPort_address(object sender, EventArgs e)
622        {
623            WPort.address();
624        }
625
626        private void WPort_data(object sender, EventArgs e)
627        {
628            WPort.data();
629        }
630
631        private void WPort_close(object sender, EventArgs e)
632        {
633            WPort.close();
634        }
635
636        private void WPort_sendFrame(object sender, EventArgs e)
637        {
638            WPort.sendFrame();
639        }
640
641        private void WPort_sendMessage(object sender, EventArgs e)
642        {
643            WPort.sendMessage();
644        }
645
646        private void WPort_received(object sender, EventArgs e)
647        {
648            WPort.received();
649        }
650
651        private void WPort_address(object sender, EventArgs e)
652        {
653            WPort.address();
654        }
655
656        private void WPort_data(object sender, EventArgs e)
657        {
658            WPort.data();
659        }
660
661        private void WPort_close(object sender, EventArgs e)
662        {
663            WPort.close();
664        }
665
666        private void WPort_sendFrame(object sender, EventArgs e)
667        {
668            WPort.sendFrame();
669        }
670
671        private void WPort_sendMessage(object sender, EventArgs e)
672        {
673            WPort.sendMessage();
674        }
675
676        private void WPort_received(object sender, EventArgs e)
677        {
678            WPort.received();
679        }
680
681        private void WPort_address(object sender, EventArgs e)
682        {
683            WPort.address();
684        }
685
686        private void WPort_data(object sender, EventArgs e)
687        {
688            WPort.data();
689        }
690
691        private void WPort_close(object sender, EventArgs e)
692        {
693            WPort.close();
694        }
695
696        private void WPort_sendFrame(object sender, EventArgs e)
697        {
698            WPort.sendFrame();
699        }
700
701        private void WPort_sendMessage(object sender, EventArgs e)
702        {
703            WPort.sendMessage();
704        }
705
706        private void WPort_received(object sender, EventArgs e)
707        {
708            WPort.received();
709        }
710
711        private void WPort_address(object sender, EventArgs e)
712        {
713            WPort.address();
714        }
715
716        private void WPort_data(object sender, EventArgs e)
717        {
718            WPort.data();
719        }
720
721        private void WPort_close(object sender, EventArgs e)
722        {
723            WPort.close();
724        }
725
726        private void WPort_sendFrame(object sender, EventArgs e)
727        {
728            WPort.sendFrame();
729        }
730
731        private void WPort_sendMessage(object sender, EventArgs e)
732        {
733            WPort.sendMessage();
734        }
735
736        private void WPort_received(object sender, EventArgs e)
737        {
738            WPort.received();
739        }
740
741        private void WPort_address(object sender, EventArgs e)
742        {
743            WPort.address();
744        }
745
746        private void WPort_data(object sender, EventArgs e)
747        {
748            WPort.data();
749        }
750
751        private void WPort_close(object sender, EventArgs e)
752        {
753            WPort.close();
754        }
755
756        private void WPort_sendFrame(object sender, EventArgs e)
757        {
758            WPort.sendFrame();
759        }
760
761        private void WPort_sendMessage(object sender, EventArgs e)
762        {
763            WPort.sendMessage();
764        }
765
766        private void WPort_received(object sender, EventArgs e)
767        {
768            WPort.received();
769        }
770
771        private void WPort_address(object sender, EventArgs e)
772        {
773            WPort.address();
774        }
775
776        private void WPort_data(object sender, EventArgs e)
777        {
778            WPort.data();
779        }
780
781        private void WPort_close(object sender, EventArgs e)
782        {
783            WPort.close();
784        }
785
786        private void WPort_sendFrame(object sender, EventArgs e)
787        {
788            WPort.sendFrame();
789        }
790
791        private void WPort_sendMessage(object sender, EventArgs e)
792        {
793            WPort.sendMessage();
794        }
795
796        private void WPort_received(object sender, EventArgs e)
797        {
798            WPort.received();
799        }
800
801        private void WPort_address(object sender, EventArgs e)
802        {
803            WPort.address();
804        }
805
806        private void WPort_data(object sender, EventArgs e)
807        {
808            WPort.data();
809        }
810
811        private void WPort_close(object sender, EventArgs e)
812        {
813            WPort.close();
814        }
815
816        private void WPort_sendFrame(object sender, EventArgs e)
817        {
818            WPort.sendFrame();
819        }
820
821        private void WPort_sendMessage(object sender, EventArgs e)
822        {
823            WPort.sendMessage();
824        }
825
826        private void WPort_received(object sender, EventArgs e)
827        {
828            WPort.received();
829        }
830
831        private void WPort_address(object sender, EventArgs e)
832        {
833            WPort.address();
834        }
835
836        private void WPort_data(object sender, EventArgs e)
837        {
838            WPort.data();
839        }
840
841        private void WPort_close(object sender, EventArgs e)
842        {
843            WPort.close();
844        }
845
846        private void WPort_sendFrame(object sender, EventArgs e)
847        {
848            WPort.sendFrame();
849        }
850
851        private void WPort_sendMessage(object sender, EventArgs e)
852        {
853            WPort.sendMessage();
854        }
855
856        private void WPort_received(object sender, EventArgs e)
857        {
858            WPort.received();
859        }
860
861        private void WPort_address(object sender, EventArgs e)
862        {
863            WPort.address();
864        }
865
866        private void WPort_data(object sender, EventArgs e)
867        {
868            WPort.data();
869        }
870
871        private void WPort_close(object sender, EventArgs e)
872        {
873            WPort.close();
874        }
875
876        private void WPort_sendFrame(object sender, EventArgs e)
877        {
878            WPort.sendFrame();
879        }
880
881        private void WPort_sendMessage(object sender, EventArgs e)
882        {
883            WPort.sendMessage();
884        }
885
886        private void WPort_received(object sender, EventArgs e)
887        {
888            WPort.received();
889        }
890
891        private void WPort_address(object sender, EventArgs e)
892        {
893            WPort.address();
894        }
895
896        private void WPort_data(object sender, EventArgs e)
897        {
898            WPort.data();
899        }
900
901        private void WPort_close(object sender, EventArgs e)
902        {
903            WPort.close();
904        }
905
906        private void WPort_sendFrame(object sender, EventArgs e)
907        {
908            WPort.sendFrame();
909        }
910
911        private void WPort_sendMessage(object sender, EventArgs e)
912        {
913            WPort.sendMessage();
914        }
915
916        private void WPort_received(object sender, EventArgs e)
917        {
918            WPort.received();
919        }
920
921        private void WPort_address(object sender, EventArgs e)
922        {
923            WPort.address();
924        }
925
926        private void WPort_data(object sender, EventArgs e)
927        {
928            WPort.data();
929        }
930
931        private void WPort_close(object sender, EventArgs e)
932        {
933            WPort.close();
934        }
935
936        private void WPort_sendFrame(object sender, EventArgs e)
937        {
938            WPort.sendFrame();
939        }
940
941        private void WPort_sendMessage(object sender, EventArgs e)
942        {
943            WPort.sendMessage();
944        }
945
946        private void WPort_received(object sender, EventArgs e)
947        {
948            WPort.received();
949        }
950
951        private void WPort_address(object sender, EventArgs e)
952        {
953            WPort.address();
954        }
955
956        private void WPort_data(object sender, EventArgs e)
957        {
958            WPort.data();
959        }
960
961        private void WPort_close(object sender, EventArgs e)
962        {
963            WPort.close();
964        }
965
966        private void WPort_sendFrame(object sender, EventArgs e)
967        {
968            WPort.sendFrame();
969        }
970
971        private void WPort_sendMessage(object sender, EventArgs e)
972        {
973            WPort.sendMessage();
974        }
975
976        private void WPort_received(object sender, EventArgs e)
977        {
978            WPort.received();
979        }
980
981        private void WPort_address(object sender, EventArgs e)
982        {
983            WPort.address();
984        }
985
986        private void WPort_data(object sender, EventArgs e)
987        {
988            WPort.data();
989        }
990
991        private void WPort_close(object sender, EventArgs e)
992        {
993            WPort.close();
994        }
995
996        private void WPort_sendFrame(object sender, EventArgs e)
997        {
998            WPort.sendFrame();
999        }
1000
1001        private void WPort_sendMessage(object sender, EventArgs e)
1002        {
1003            WPort.sendMessage();
1004        }
1005
1006        private void WPort_received(object sender, EventArgs e)
1007        {
1008            WPort.received();
1009        }
1010
1011        private void WPort_address(object sender, EventArgs e)
1012        {
1013            WPort.address();
1014        }
1015
1016        private void WPort_data(object sender, EventArgs e)
1017        {
1018            WPort.data();
1019        }
1020
1021        private void WPort_close(object sender, EventArgs e)
1022        {
1023            WPort.close();
1024        }
1025
1026        private void WPort_sendFrame(object sender, EventArgs e)
1027        {
1028            WPort.sendFrame();
1029        }
1030
1031        private void WPort_sendMessage(object sender, EventArgs e)
1032        {
1033            WPort.sendMessage();
1034        }
1035
1036        private void WPort_received(object sender, EventArgs e)
1037        {
1038            WPort.received();
1039        }
1040
1041        private void WPort_address(object sender, EventArgs e)
1042        {
1043            WPort.address();
1044        }
1045
1046        private void WPort_data(object sender, EventArgs e)
1047        {
1048            WPort.data();
1049        }
1050
1051        private void WPort_close(object sender, EventArgs e)
1052        {
1053            WPort.close();
1054        }
1055
1056        private void WPort_sendFrame(object sender, EventArgs e)
1057        {
1058            WPort.sendFrame();
1059        }
1060
1061        private void WPort_sendMessage(object sender, EventArgs e)
1062        {
1063            WPort.sendMessage();
1064        }
1065
1066        private void WPort_received(object sender, EventArgs e)
1067        {
1068            WPort.received();
1069        }
1070
1071        private void WPort_address(object sender, EventArgs e)
1072        {
1073            WPort.address();
1074        }
1075
1076        private void WPort_data(object sender, EventArgs e)
1077        {
1078            WPort.data();
1079        }
1080
1081        private void WPort_close(object sender, EventArgs e)
1082        {
1083            WPort.close();
1084        }
1085
1086        private void WPort_sendFrame(object sender, EventArgs e)
1087        {
1088            WPort.sendFrame();
1089        }
1090
1091        private void WPort_sendMessage(object sender, EventArgs e)
1092        {
1093            WPort.sendMessage();
1094        }
1095
1096        private void WPort_received(object sender, EventArgs e)
1097        {
1098            WPort.received();
1099        }
1100
1101        private void WPort_address(object sender, EventArgs e)
1102        {
1103            WPort.address();
1104        }
1105
1106        private void WPort_data(object sender, EventArgs e)
1107        {
1108            WPort.data();
1109        }
1110
1111        private void WPort_close(object sender, EventArgs e)
1112        {
1113            WPort.close();
1114        }
1115
1116        private void WPort_sendFrame(object sender, EventArgs e)
1117        {
1118            WPort.sendFrame();
1119        }
1120
1121        private void WPort_sendMessage(object sender, EventArgs e)
1122        {
1123            WPort.sendMessage();
1124        }
1125
1126        private void WPort_received(object sender, EventArgs e)
1127        {
1128            WPort.received();
1129        }
1130
1131        private void WPort_address(object sender, EventArgs e)
1132        {
1133            WPort.address();
1134        }
1135
1136        private void WPort_data(object sender, EventArgs e)
1137        {
1138            WPort.data();
1139        }
1140
1141        private void WPort_close(object sender, EventArgs e)
1142        {
1143            WPort.close();
1144        }
1145
1146        private void WPort_sendFrame(object sender, EventArgs e)
1147        {
1148            WPort.sendFrame();
1149        }
1150
1151        private void WPort_sendMessage(object sender, EventArgs e)
1152        {
1153            WPort.sendMessage();
1154        }
1155
1156        private void WPort_received(object sender, EventArgs e)
1157        {
1158            WPort.received();
1159        }
1160
1161        private void WPort_address(object sender, EventArgs e)
1162        {
1163            WPort.address();
1164        }
1165
1166        private void WPort_data(object sender, EventArgs e)
1167        {
1168            WPort.data();
1169        }
1170
1171        private void WPort_close(object sender, EventArgs e)
1172        {
1173            WPort.close();
1174        }
1175
1176        private void WPort_sendFrame(object sender, EventArgs e)
1177        {
1178            WPort.sendFrame();
1179        }
1180
1181        private void WPort_sendMessage(object sender, EventArgs e)
1182        {
1183            WPort.sendMessage();
1184        }
1185
1186        private void WPort_received(object sender, EventArgs e)
1187        {
1188            WPort.received();
1189        }
1190
1191        private void WPort_address(object sender, EventArgs e)
1192        {
1193            WPort.address();
1194        }
1195
1196        private void WPort_data(object sender, EventArgs e)
1197        {
1198            WPort.data();
1199        }
1200
1201        private void WPort_close(object sender, EventArgs e)
1202        {
1203            WPort.close();
1204        }
1205
1206        private void WPort_sendFrame(object sender, EventArgs e)
1207        {
1208            WPort.sendFrame();
1209        }
1210
1211        private void WPort_sendMessage(object sender, EventArgs e)
1212        {
1213            WPort.sendMessage();
1214        }
1215
1216        private void WPort_received(object sender, EventArgs e)
1217        {
1218            WPort.received();
1219        }
1220
1221        private void WPort_address(object sender, EventArgs e)
1222        {
1223            WPort.address();
1224        }
1225
1226        private void WPort_data(object sender, EventArgs e)
1227        {
1228            WPort.data();
1229        }
1230
1231        private void WPort_close(object sender, EventArgs e)
1232        {
1233            WPort.close();
1234        }
1235
1236        private void WPort_sendFrame(object sender, EventArgs e)
1237        {
1238            WPort.sendFrame();
1239        }
1240
1241        private void WPort_sendMessage(object sender, EventArgs e)
1242        {
1243            WPort.sendMessage();
1244        }
1245
1246        private void WPort_received(object sender, EventArgs e)
1247        {
1248            WPort.received();
1249        }
1250
1251        private void WPort_address(object sender, EventArgs e)
1252        {
1253            WPort.address();
1254        }
1255
1256        private void WPort_data(object sender, EventArgs e)
1257        {
1258            WPort.data();
1259        }
1260
1261        private void WPort_close(object sender, EventArgs e)
1262        {
1263            WPort.close();
1264        }
1265
1266        private void WPort_sendFrame(object sender, EventArgs e)
1267        {
1268            WPort.sendFrame();
1269        }
1270
1271        private void WPort_sendMessage(object sender, EventArgs e)
1272        {
1273            WPort.sendMessage();
1274        }
1275
1276        private void WPort_received(object sender, EventArgs e)
1277        {
1278            WPort.received();
1279        }
1280
1281        private void WPort_address(object sender, EventArgs e)
1282        {
1283            WPort.address();
1284        }
1285
1286        private void WPort_data(object sender, EventArgs e)
1287        {
1288            WPort.data();
1289        }
1290
1291        private void WPort_close(object sender, EventArgs e)
1292        {
1293            WPort.close();
1294        }
1295
1296        private void WPort_sendFrame(object sender, EventArgs e)
1297        {
1298            WPort.sendFrame();
1299        }
1300
1301        private void WPort_sendMessage(object sender, EventArgs e)
1302        {
1303            WPort.sendMessage();
1304        }
1305
1306        private void WPort_received(object sender, EventArgs e)
1307        {
1308            WPort.received();
1309        }
1310
1311        private void WPort_address(object sender, EventArgs e)
1312        {
1313            WPort.address();
1314        }
1315
1316        private void WPort
```

```

46 }
47
48     selCom.DataSource=comList;
49     selCom.SelectedIndex=0;
50 }
51
52 public void alarmRang(object sender, Wavenis.IWavenisEvent e)
53 {
54     try
55     {
56         // Print the alarm = "address => value"
57         received.Text = e.GetAddress() + "=" + e.GetValue();
58         // Do all events and then send the ResetAlarm function
59         Application.DoEvents();
60         string error=null;
61         string rep=null;
62         this.WPort.ResetAlarm( e.GetAddress(),rep,e.GetStatus(),ref error);
63     }
64     catch(Exception ex)
65     {
66         received.Text = "alarm exception=" + ex.Message;
67     }
68 }

69
70 /// <summary>
71 /// Nettoyage des ressources utilisées.
72 /// </summary>
73 protected override void Dispose( bool disposing )
74 {
75     if( disposing )
76     {
77         if (components != null)
78         {
79             components.Dispose();
80         }
81     }
82     // Close the Application ressources
83     if(WPort!=null)
84         WPort.Close();

85
86     base.Dispose( disposing );
87 }

88
89 #region Code généré par le Concepteur Windows Form
90 /// <summary>
91 /// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
92 /// le contenu de cette méthode avec l'éditeur de code.
93 /// </summary>
94 private void InitializeComponent()

```

```
95      {
96          this.selCom = new System.Windows.Forms.ComboBox();
97          this.open = new System.Windows.Forms.Button();
98          this.sendMessage = new System.Windows.Forms.Button();
99          this.sendFrame = new System.Windows.Forms.Button();
100         this.address = new System.Windows.Forms.TextBox();
101         this.received = new System.Windows.Forms.Label();
102         this.data = new System.Windows.Forms.TextBox();
103         this.SuspendLayout();
104         //
105         // selCom
106         //
107         this.selCom.Location = new System.Drawing.Point(24, 16);
108         this.selCom.Name = "selCom";
109         this.selCom.Size = new System.Drawing.Size(64, 21);
110         this.selCom.TabIndex = 0;
111         //
112         // open
113         //
114         this.open.Location = new System.Drawing.Point(120, 16);
115         this.open.Name = "open";
116         this.open.Size = new System.Drawing.Size(40, 24);
117         this.open.TabIndex = 1;
118         this.open.Text = "Open";
119         this.open.Click += new System.EventHandler(this.open_Click);
120         //
121         // sendMessage
122         //
123         this.sendMessage.Location = new System.Drawing.Point(192, 48);
124         this.sendMessage.Name = "sendMessage";
125         this.sendMessage.Size = new System.Drawing.Size(88, 24);
126         this.sendMessage.TabIndex = 2;
127         this.sendMessage.Text = "SendMessage";
128         this.sendMessage.Click += new System.EventHandler(this.sendMessage_Click);
129         //
130         // sendFrame
131         //
132         this.sendFrame.Location = new System.Drawing.Point(120, 48);
133         this.sendFrame.Name = "sendFrame";
134         this.sendFrame.Size = new System.Drawing.Size(72, 24);
135         this.sendFrame.TabIndex = 3;
136         this.sendFrame.Text = "SendFrame";
137         this.sendFrame.Click += new System.EventHandler(this.sendFrame_Click);
138         //
139         // address
140         //
141         this.address.Location = new System.Drawing.Point(24, 48);
142         this.address.Name = "address";
143         this.address.Size = new System.Drawing.Size(88, 20);
```

```

144         this.address.TabIndex = 4;
145         this.address.Text = "011603300373";
146         //
147         // received
148         //
149         this.received.Location = new System.Drawing.Point(24, 112);
150         this.received.Name = "received";
151         this.received.Size = new System.Drawing.Size(256, 152);
152         this.received.TabIndex = 5;
153         //
154         // data
155         //
156         this.data.Location = new System.Drawing.Point(24, 80);
157         this.data.Name = "data";
158         this.data.Size = new System.Drawing.Size(256, 20);
159         this.data.TabIndex = 6;
160         this.data.Text = "01";
161         //
162         // Form1
163         //
164         this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
165         this.ClientSize = new System.Drawing.Size(292, 273);
166         this.Controls.Add(this.data);
167         this.Controls.Add(this.received);
168         this.Controls.Add(this.address);
169         this.Controls.Add(this.sendFrame);
170         this.Controls.Add(this.sendMessage);
171         this.Controls.Add(this.open);
172         this.Controls.Add(this.selCom);
173         this.Name = "Form1";
174         this.Text = "Form1";
175         this.ResumeLayout(false);
176     }
177 #endregion
178
179 /// <summary>
180 /// Point d'entrée principal de l'application.
181 /// </summary>
182 [STAThread]
183 static void Main()
184 {
185     Application.Run(new Form1());
186 }
187
188 private void open_Click(object sender, System.EventArgs e)
189 {
190     short sCom = 0;
191     sCom = (short)selCom.SelectedIndex;
192     sCom++;

```

```

193         // Com Port selection
194         WPort.SetCommPortNumber(sCom);
195
196
197         // WavePort Initialization
198         bool bresult = WPort.Init(9600,8,0,1,0);
199         if(true!=bresult)
200             received.Text = "NOK";
201         else
202         {
203             received.Text = "OK";
204             // If result successed, set up the event alarm handler
205             WPort.Alarm += new Wavenis.alarmHandler(alarmRang);
206         }
207     }
208
209     private void sendFrame_Click(object sender, System.EventArgs e)
210     {
211         string error=null;
212         int timeout=6100;
213         string repeater=null;
214         // Send the frame and wait for the response
215         string result = WPort.RadioExchange(address.Text,data.Text,repeater,timeout,ref error);
216         if(null!=result)
217         {
218             // Print the response
219             received.Text = result;
220         }
221         else
222         {
223             //Print the error if the function failed
224             received.Text = error;
225         }
226     }
227
228     private void sendMessage_Click(object sender, System.EventArgs e)
229     {
230         string error=null;
231         string repeater=null;
232         string result = WPort.RadioMessage(address.Text,data.Text,repeater,ref error);
233         if(null!=result)
234         {
235             // Print the response but no response with the RadioMessage function...
236             received.Text = result;
237         }
238         else
239         {
240             //Print the error if the function failed
241             received.Text = error;

```

242		}
243		}
244		}
245	}	

APPENDIX 5 – C# .NET EXAMPLE CODE FOR WINCE

```
1 using System;
2 using System.Drawing;
3 using System.Collections;
4 using System.Windows.Forms;
5 using System.Data;
6 using Wavenis; // Add Wavenis reference
7
8 namespace WinCEWPortCSharp
9 {
10     /// <summary>
11     /// Description résumée de Form1.
12     /// </summary>
13     public class Form1 : System.Windows.Forms.Form
14     {
15         private System.Windows.Forms.TextBox data;
16         private System.Windows.Forms.Label received;
17         private System.Windows.Forms.TextBox address;
18         private System.Windows.Forms.Button sendFrame;
19         private System.Windows.Forms.Button sendMessage;
20         private System.Windows.Forms.Button open;
21         private System.Windows.Forms.ComboBox selCom;
22         private System.Windows.Forms.Button quit;
23         private Wavenis.WavePort WPort;
24
25         public Form1()
26         {
27             //
28             // Requis pour la prise en charge du Concepteur Windows Forms
29             //
30             InitializeComponent();
31
32             //
33             // TODO : ajoutez le code du constructeur après l'appel à InitializeComponent
34             //
35
36             // Create instance
37             WPort = new WavePort();
38
39             // Form initialization
40             string[] comList = new string[9];
41             for (int nIndex=1; nIndex<10; nIndex++)
42             {
43                 string text= "COM" + nIndex.ToString() + ":";
44                 comList[nIndex-1]=text;
45             }
46             selCom.DataSource=comList;
```

```

47             selCom.SelectedIndex=0;
48         }
49     public void alarmRang(object sender, Wavenis.IWavenisEvent e)
50     {
51         try
52         {
53             // Print the alarm = "address => value"
54             received.Text = e.GetAddress() + "=" + e.GetValue();
55             // Do all events and then send the ResetAlarm function
56             Application.DoEvents();
57             string error=null;
58             string rep=null;
59             this.WPort.ResetAlarm( e.GetAddress(),rep,e.GetStatus(),ref error);
60         }
61         catch(Exception ex)
62         {
63             received.Text = "alarm exception=" + ex.Message;
64         }
65     }
66
67 /// <summary>
68 /// Nettoyage des ressources utilisées.
69 /// </summary>
70 protected override void Dispose( bool disposing )
71 {
72     // Close the Application ressources if necessary
73     if(WPort!=null)
74         WPort.Close();
75
76     base.Dispose( disposing );
77 }
78
79 #region Code généré par le Concepteur Windows Form
80 /// <summary>
81 /// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
82 /// le contenu de cette méthode avec l'éditeur de code.
83 /// </summary>
84 private void InitializeComponent()
85 {
86     this.data = new System.Windows.Forms.TextBox();
87     this.received = new System.Windows.Forms.Label();
88     this.address = new System.Windows.Forms.TextBox();
89     this.sendFrame = new System.Windows.Forms.Button();
90     this.sendMessage = new System.Windows.Forms.Button();
91     this.open = new System.Windows.Forms.Button();
92     this.selCom = new System.Windows.Forms.ComboBox();
93     this.quit = new System.Windows.Forms.Button();
94
95     // data

```

```
96 //  
97 this.data.Location = new System.Drawing.Point(16, 72);  
98 this.data.Size = new System.Drawing.Size(256, 20);  
99 this.data.Text = "01";  
100 //  
101 // received  
102 //  
103 this.received.Location = new System.Drawing.Point(16, 104);  
104 this.received.Size = new System.Drawing.Size(256, 56);  
105 //  
106 // address  
107 //  
108 this.address.Location = new System.Drawing.Point(16, 40);  
109 this.address.Size = new System.Drawing.Size(96, 20);  
110 this.address.Text = "011603300373";  
111 //  
112 // sendFrame  
113 //  
114 this.sendFrame.Location = new System.Drawing.Point(112, 40);  
115 this.sendFrame.Size = new System.Drawing.Size(72, 24);  
116 this.sendFrame.Text = "SendFrame";  
117 this.sendFrame.Click += new System.EventHandler(this.sendFrame_Click);  
118 //  
119 // sendMessage  
120 //  
121 this.sendMessage.Location = new System.Drawing.Point(184, 40);  
122 this.sendMessage.Size = new System.Drawing.Size(88, 24);  
123 this.sendMessage.Text = "SendMessage";  
124 this.sendMessage.Click += new System.EventHandler(this.sendMessage_Click);  
125 //  
126 // open  
127 //  
128 this.open.Location = new System.Drawing.Point(112, 8);  
129 this.open.Size = new System.Drawing.Size(40, 24);  
130 this.open.Text = "Open";  
131 this.open.Click += new System.EventHandler(this.open_Click);  
132 //  
133 // selCom  
134 //  
135 this.selCom.Location = new System.Drawing.Point(16, 8);  
136 this.selCom.Size = new System.Drawing.Size(72, 21);  
137 //  
138 // quit  
139 //  
140 this.quit.Location = new System.Drawing.Point(232, 8);  
141 this.quit.Size = new System.Drawing.Size(40, 24);  
142 this.quit.Text = "Quit";  
143 this.quit.Click += new System.EventHandler(this.quit_Click);  
144 //
```

```

145         // Form1
146         //
147         this.ClientSize = new System.Drawing.Size(282, 167);
148         this.ControlBox = false;
149         this.Controls.Add(this.quit);
150         this.Controls.Add(this.data);
151         this.Controls.Add(this.received);
152         this.Controls.Add(this.address);
153         this.Controls.Add(this.sendFrame);
154         this.Controls.Add(this.sendMessage);
155         this.Controls.Add(this.open);
156         this.Controls.Add(this.selCom);
157         this.Text = "Form1";
158     }
159 #endregion
160
161 /// <summary>
162 /// Point d'entrée principal de l'application.
163 /// </summary>
164 static void Main()
165 {
166     Application.Run(new Form1());
167 }
168
169 private void open_Click(object sender, System.EventArgs e)
170 {
171     short sCom = 0;
172     sCom = (short)selCom.SelectedIndex;
173     sCom++;
174
175     // Com Port selection
176     WPort.SetCommPortNumber(sCom);
177
178     // WavePort Initialization
179     bool bresult = WPort.Init(9600,8,0,1,0);
180
181     if(true!=bresult)
182         received.Text = "NOK";
183     else
184     {
185         received.Text = "OK";
186         // If result successed, set up the event alarm handler
187         WPort.Alarm += new Wavenis.alarmHandler(alarmRang);
188     }
189 }
190
191
192
193 private void sendFrame_Click(object sender, System.EventArgs e)

```

```
194    {
195        string error=null;
196        int timeout=6100;
197        string repeater=null;
198        // Send the frame and wait for the response
199        string result = WPort.RadioExchange(address.Text,data.Text,repeater,timeout,ref error);
200        if(null!=result)
201        {
202            // Print the response
203            received.Text = result;
204        }
205        else
206        {
207            //Print the error if the function failed
208            received.Text = error;
209        }
210    }
211
212    private void sendMessage_Click(object sender, System.EventArgs e)
213    {
214        string error=null;
215        string repeater=null;
216        // Send message
217        string result = WPort.RadioMessage(address.Text,data.Text,repeater,ref error);
218
219        if(null!=result)
220        {
221            // Print the response but no response with the RadioMessage function...
222            received.Text = result;
223        }
224        else
225        {
226            //Print the error if the function failed
227            received.Text = error;
228        }
229    }
230
231    private void quit_Click(object sender, System.EventArgs e)
232    {
233        // Close the Application ressources
234        if(WPort!=null)
235            WPort.Close();
236        Application.Exit();
237    }
238}
239}
```